Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

APLICACIÓN DE LA METODOLOGÍA DE LA EMPRESA TATA CONSULTANCY SERVICES A LAS ÁREAS DE PROGRAMACIÓN, REDES Y BASES DE DATOS DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS, DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Mario José Bautista Fuentes
Asesorado por el Ing. Jorge Armín Mazariegos

Guatemala, octubre de 2009

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

APLICACIÓN DE LA METODOLOGÍA DE LA EMPRESA TATA CONSULTANCY SERVICES A LAS ÁREAS DE PROGRAMACIÓN, REDES Y BASES DE DATOS DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS, DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

TRABAJO DE GRADUACIÓN DE EPS

PRESENTADO A LA JUNTA DIRECTIVA
DE LA FACULTAD DE INGENIERÍA
POR

MARIO JOSÉ BAUTISTA FUENTES

ASESORADO POR EL ING. JORGE ARMÍN MAZARIEGOS
AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 2009

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Inga. Glenda Patricia García Soria
VOCAL II	Inga. Alba Maritza Guerrero de López
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. José Milton De León Bran
VOCAL V	Br. Isaac Sultán Mejía
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADORA	Inga. Floriza Felipa Ávila Pesquera
EXAMINADOR	Ing. Marlon Antonio Pérez Turk
EXAMINADORA	Inga. Sonia Yolanda Castañeda Ramírez
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de Ejercicio Profesional Supervisado (EPS) titulado:

APLICACIÓN DE LA METODOLOGÍA DE LA EMPRESA TATA CONSULTANCY SERVICES A LAS ÁREAS DE PROGRAMACIÓN, REDES Y BASES DE DATOS DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS, DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA,

tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, en agosto 2008.

Mario José Bautista Fuentes.

Guatemala, 29 de agosto 2009

Ingeniera
Norma Ileana Sarmiento
Directora Unidad EPS
Dirección de EPS
Facultad de Ingeniería
USAC

Respetable Ingeniera Sarmiento:

Por este medio hado de su conocimiento que he revisado el trabajo de graduación del estudiante **MARIO JOSÉ BAUTISTA FUENTES**, titulado: "**Aplicación de la metodología de la empresa TATA CONSULTANCY SERVICES a las áreas de Programación, Redes y Bases de Datos de la carrera de Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala**", y a mi criterio, el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Agradeciendo su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Jorge Armin Mazariegos
Catedrático
Asesor del Proyecto
Escuela de Ciencias y Sistemas

Universidad de San Carlos de Guatemala
Facultad de Ingeniería

UNIDAD DE E.P.S.

Guatemala, 03 de septiembre de 2009.
REF.EPS.DOC.1296.09.09.

Inga. Norma Ileana Sarmiento Zeceña de Serrano
Directora Unidad de EPS
Facultad de Ingeniería
Presente

Estimada Ingeniera Sarmiento Zeceña.

Por este medio atentamente le informo que como Supervisora de la Práctica del Ejercicio Profesional Supervisado, (E.P.S) del estudiante universitario de la Carrera de Ingeniería en Ciencias y Sistemas, **Mario José Bautista Fuentes** Carné No. **200312942** procedí a revisar el informe final, cuyo título es **"APLICACIÓN DE LA METODOLOGÍA DE LA EMPRESA TATA CONSULTANCY SERVICES A LAS ÁREAS DE PROGRAMACIÓN, REDES Y BASES DE DATOS DE LA CARRERA DE CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA".**

En tal virtud, **LO DOY POR APROBADO**, solicitándole darle el trámite respectivo.

Sin otro particular, me es grato suscribirme.

Atentamente,

"Id y Enseñad a Todos"

Inga. Floriza Felipa Ávila Pesquera de Medinilla
Supervisora de EPS
Área de Ingeniería en Ciencias y Sistemas

Universidad de San Carlos de Guatemala
SUPERVISOR (A) DE EPS
Unidad de Prácticas de Ingeniería y EPS
Facultad de Ingeniería

FFAPdM/RA

Universidad de San Carlos de Guatemala
Facultad de Ingeniería

UNIDAD DE E.P.S.

Guatemala, 03 de septiembre de 2009.
REF.EPS.D.545.09.09.

Ing. Marlon Antonio Pérez Turck
Director Escuela de Ingeniería Ciencias y Sistemas
Facultad de Ingeniería
Presente

Estimado Ingeniero Perez Turck.

Por este medio atentamente le envío el informe final correspondiente a la práctica del Ejercicio Profesional Supervisado, (E.P.S) titulado **"APLICACIÓN DE LA METODOLOGÍA DE LA EMPRESA TATA CONSULTANCY SERVICES A LAS ÁREAS DE PROGRAMACIÓN, REDES Y BASES DE DATOS DE LA CARRERA DE CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA",** que fue desarrollado por el estudiante universitario **Mario José Bautista Fuentes** Carné No. **200312942** quien fue debidamente asesorado por el Ing. **Jorge Armin Mazariegos** y supervisado por la Inga. Floriza Felipa Ávila Pesquera de Medinilla

Por lo que habiendo cumplido con los objetivos y requisitos de ley del referido trabajo y existiendo la aprobación del mismo por parte del Asesor y de la Supervisora de EPS, en mi calidad de Directora apruebo su contenido solicitándole darle el trámite respectivo.

Sin otro particular, me es grato suscribirme.

Atentamente,
"Id y Enseñad a Todos"

Inga. Norma Ileana Sarmiento Zeceña de Serrano
Directora Unidad de EPS
DIRECCION
Unidad de Prácticas de Ingeniería y EPS
Facultad de Ingeniería

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 09 de Septiembre de 2009

Ingeniero
Marlon Antonio Pérez Turk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **MARIO JOSE BAUTISTA FUENTES,** titulado: **"APLICACIÓN DE LA METODOLOGIA DE LA EMPRESA TATA CONSULTANCY SERVICES A LAS AREAS DE PROGRAMACION, REDES Y BASES DE DATOS DE LA CARRERA DE CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERIA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA"**, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

ESCUELA DE CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, de trabajo de graduación titulado ***"APLICACIÓN DE LA METODOLOGÍA DE LA EMPRESA TATA CONSULTANCY SERVICES A LAS ÁREAS DE PROGRAMACIÓN, REDES Y BASES DE DATOS DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA."****, presentado por el estudiante MARIO JOSÉ BAUTISTA FUENTES, aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

Ing. Marlon Antonio Pérez Turk
Director, Escuela de Ingeniería Ciencias y Sistemas

Guatemala, 05 de octubre 2009

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.372.09

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la áprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado: **APLICACIÓN DE LA METODOLOGÍA DE LA EMPRESA TATA CONSULTANCY SERVICES A LAS ÁREAS DE PROGRAMACIÓN, REDES Y BASES DE DATOS DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS, DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA,** presentado por el estudiante universitario **Mario José Bautista Fuentes,** procede a la autorización para la impresión del mismo.

IMPRÍMASE.

Ing. Murphy Olympo Paiz Recinos
DECANO

Guatemala, octubre de 2009

ACTO QUE DEDICO A:

DIOS y Santa María Virgen	Por concederme el privilegio de culminar mi carrera universitaria.
Mis padres:	Mario Gabriel Bautista Monzón e Irma Esperanza Fuentes Laparra, por sus sabios consejos, abnegado esfuerzo al guiarme por el camino del bien y su apoyo en todo momento a lo largo de mi carrera. Ellos son los que me han impulsado y motivado en todo momento para alcanzar este triunfo.

AGRADECIMIENTOS A:

Mi asesor:	Ingeniero Jorge Armín Mazariegos, por su apoyo, guía y tiempo dedicado en la elaboración de este trabajo.
Universidad de San Carlos, Facultad de Ingeniería:	Por la formación recibida durante la carrera, especialmente la recibida por parte del personal docente de la Escuela de Ciencias y Sistemas.
Mis supervisores de EPS:	Los ingenieros: Marlon Turk, Luis Espino, Pedro Hernández y Floriza Ávila.
Mi hermano:	José Gabriel Bautista Fuentes por su apoyo y confianza demostrada siempre.
Mis amigos:	Especialmente a Marlon Palma, Juan Barillas, Haroldo Rojas, Framklym Algaba, Rafael Siney, Rodolfo Zea, René Chinchilla, Raúl Huertas y Francisco Cruz.
Mis amigas:	Especialmente a Heidy Bonilla y Luz Arrecis.
Mis tíos:	Especialmente a Helen Judith y Osberto Gonzalo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

EPS	Ejercicio Profesional Supervisado
Instructor Guideline	Un IG es conocido como guía del instructor. Esta guía define un programa detallado de un curso específico que puede ser utilizado por catedráticos y auxiliares como guía para impartir un curso internamente.
IPC1	Introducción a la Programación y Computación 1
IPC2	Introducción a la Programación y Computación 2
Módulo	Es una unidad por medio de la cual se encuentra integrado un curso específico.
NGN	*Next Generation Networks*. Redes de Nueva Generación.

Sesión	Definida desde el punto de vista educativo, es la agrupación de temas correspondientes a un curso que serán impartidos en un espacio temporal definido.
Sub-módulo	Es un componente de un sub-módulo. Esto quiere decir que el tema principal del módulo posee sub-temas que en este caso son los sub-módulos.

RESUMEN

El mundo actualmente afronta un avance tecnológico demasiado acelerado. Este avance origina que tecnologías del tipo de la información y la comunicación se expandan y cambien de una manera inimaginable. La carrera de Ciencias y Sistemas debe ser capaz de poder afrontar estos cambios y ser capaz de mantener el ritmo de los mismos. Mantener el ritmo significa que para poder dar una enseñanza de calidad se actualicen contenidos y se realicen estrategias de enseñanza óptimas. Estas permitirán que el egresado de la carrera sea capaz de afrontar las demandas en el mercado laboral, debido a que poseerán un nivel competitivo.

La forma de afrontar estas demandas y cambios se realiza por medio de estructurar el contenido de los cursos. Esta estructuración permite fácilmente actualizar el contenido y así afrontar el avance tecnológico. El presente Ejercicio Profesional Supervisado se elaboró partiendo de una metodología que permite estructurar, organizar y definir los contenidos de los cursos. Estos se agrupan por medio de sesiones, módulos y sub-módulos, para que mediante el empleo de tecnología brindar una enseñanza eficiente y eficaz de los cursos de la carrera.

Una documentación de apoyo relacionada con cada curso, permite fundamentar la estructura de los cursos. Esta documentación se encuentra integrada por temas importantes explicados de una manera sencilla. Esto es debido a que los estudiantes deben ser capaces de entender los contenidos. La

documentación de apoyo de esta forma se vuelve un apoyo fundamental para poder impartir el curso.

El centro tecnológico de la India, cuya sede se encuentra en el campus universitario, brindó apoyo al presente Ejercicio Profesional Supervisado. El apoyo de las personas de este centro permitió depurar el proceso definido de estructuración de los cursos. Esto se realizó mediante el uso de la aplicación de las experiencias en capacitación obtenidas por la empresa transnacional TATA *Consultancy Services*.

La capacitación fue realizada en el período comprendido entre los meses de enero a abril del año 2008. El proceso de creación de la guía del instructor y la documentación de apoyo de los contenidos de los cursos fue realizado posteriormente de forma individual. Estos fueron supervisados por catedráticos con experiencia en las áreas del presente trabajo. Ellos fueron asignados por el asesor del ejercicio profesional supervisado y conocen ampliamente las necesidades que afrontan los estudiantes en dichas áreas.

El presente trabajo aplicará la metodología a los cursos: Introducción a la Programación y Computación 1, Introducción a la Programación y Computación 2, Sistemas de Bases de Datos 1 y Redes de Nueva Generación.

OBJETIVOS

GENERAL

Brindar un contenido eficaz, eficiente, estructurado y estandarizado con respecto a los temas y prácticas de tres cursos que se imparten en la Escuela de Ciencias y Sistemas, utilizando la aplicación de la metodología IG (*Instructor Guideline*) para que el catedrático pueda enseñar dichos cursos, asimismo, proveer un documento integrado por los temas impartidos en cada curso para poder facilitar el proceso de enseñanza por medio de material personalizado.

ESPECÍFICOS

1. Realizar un *Instructor Guideline,* el cual permita estructurar la enseñanza, el contenido, la evaluación y prácticas de los cursos:
    - Introducción a la Programación y Computación 1.
    - Introducción a la Programación y Computación 2.
    - Redes de Nueva Generación.
2. Realizar material en forma de diapositivas, que permitan el apoyo visual del proceso de enseñanza de los cursos.
3. Realizar un libro de trabajo (*Hand Book*), que permita el establecimiento de las tareas, prácticas y proyectos, propios de los cursos.
4. Realizar documentación de apoyo, que contenga los temas propios del contenido de los cursos y de forma sencilla y práctica fomente en

los estudiantes el auto-aprendizaje y mejorar el aprendizaje de los cursos:

- Introducción a la Programación y Computación 1.
- Introducción a la Programación y Computación 2.
- Sistemas de Bases de Datos 1.

5. Brindar una estructura estándar referente a los exámenes (parciales y final) de los cursos.

.

INTRODUCCIÓN

La tecnología de la información cambia aceleradamente. Esto ocasiona que para brindar un nivel educativo óptimo, los cursos que se basan en esta tecnología deben adaptarse a los cambios constantemente. La Escuela de Ciencias y Sistemas en su esfuerzo por mantener un nivel educativo alto para los egresados de la carrera, ha optado por estandarizar de una forma estructurada los contenidos de los cursos. Esto para que los temas puedan ser adaptados constantemente a las necesidades actuales y sean de beneficio a los estudiantes.

El contenido de los cursos no es el único en ser estandarizado. El control se ha establecido para las prácticas, proyectos, tareas, y otras cosas relacionadas. Esto significa que los catedráticos poseerán una guía que les permitirá evitar traslapes de entrega de algunas de estas actividades con otros cursos.

El *India – Guatemala IT Education Center of Excellence*, ha sido un esfuerzo de la Escuela de Ciencias y Sistemas para poder mejorar la enseñanza de la carrera de Ciencias y Sistemas. Este centro permite impartir cursos certificados sobre áreas de tecnología de la información y la comunicación. El esfuerzo está enfocado en mejorar el aprendizaje tanto de los laboratorios como de los cursos de la carrera.

El *IT Center*, brindó un curso impartido por el señor Mrutunjaya Panda. El enfoque de este curso fue dar las bases para la creación de un *Instructor Guideline*. Este documento permite detallar el contenido que integra el curso,

las sesiones en las cuales se encuentra dividido, la evaluación y la lista de actividades que se desarrollan durante el curso. El IG permite entonces apoyar al catedrático al momento de impartir el curso.

El presente documento se encuentra integrado entonces por varios capítulos. Los primeros dos capítulos consisten en una introducción a la metodología de aprendizaje e IG y los elementos que integran la misma. Los tres capítulos siguientes, comprenden el diseño de las guías para los cursos de IPC1, IPC2 y NGN. Los capítulos siguientes a estos, presentan la documentación de apoyo respectiva de los cursos de IPC1, IPC2 y Sistemas de Bases de Datos 1.

1. MARCO TEÓRICO

1.1. Principios de Pedagogía

Una buena práctica pedagógica permite ampliar el potencial en el desarrollo del aprendizaje de los alumnos. La buena práctica debe ser capaz de fomentar el aprovechamiento del tiempo para enseñar, la participación proactiva de los alumnos y la retroalimentación necesaria para fomentar el aprendizaje aprovechando el uso de las diversas tecnologías.

Art Chickering y Zelda Gamson (1987, 1991) han creado los “principios de la buena práctica pedagógica”. Estos principios inicialmente fueron establecidos para la educación presencial, aunque actualmente se ha analizado su aplicación en educación a distancia. Los principios, según Chickering y Gamson, se encuentran basados en una visión de la educación como un proceso, el cual es activo, cooperativo y exigente.

Los principios propuestos por Chickering y Gamson son siete. Estos principios son:

a. Propiciar el contacto entre estudiantes y profesores. El principio se fundamenta en tratar de incrementar el contacto entre el estudiante y el profesor. Esto para involucrar y motivar al estudiante en su

aprendizaje. El conocer a su profesor empujará al estudiante a una situación, donde adquirirá un compromiso intelectual. Este compromiso lo debe motivar a pensar en sus valores, como en sus planes futuros. Al incrementar el contacto el profesor será capaz de orientar al estudiante, conforme a sus avances y que pueda sobreponerse a situaciones adversas del proceso de aprendizaje.

b. Fomentar la cooperación entre los estudiantes. Este principio se basa en el hecho de que el aprendizaje de un estudiante, aumenta cuando éste se realiza de manera conjunta con otros estudiantes. El aprendizaje bueno es social y colaborativo, y no es en ningún momento competitivo y aislado. El poder compartir las ideas propias y responder a las de los otros, mejora el pensamiento y mejora la comprensión de ideas y conceptos.
c. Propiciar el aprendizaje activo. El aprendizaje se debe experimentar, es decir participar activamente en él. Los estudiantes no deben limitarse a recibir lo que su profesor les comunica. Ellos deben ser capaces de discutir sobre su aprendizaje, deben comentarlo de forma oral o escrita, poder relacionarlo a sus experiencias y aplicarlo a sus propias vidas.
d. Proporcionar retroalimentación a tiempo. Este principio se fundamenta en que los estudiantes deben recibir la retroalimentación adecuada y en el tiempo adecuado para obtener el mejor beneficio en su aprendizaje. Esta retroalimentación permite a los estudiantes, recibir sugerencias con el fin de mejorar, además de poder determinar y evaluar su aprendizaje y su progreso.
e. Enfatizar el uso apropiado del tiempo. Las destrezas en el manejo del tiempo, son críticas tanto para los estudiantes, como para los profesionales. La planificación adecuada del contenido de aprendizaje, para que sea impartido en tiempo, además de ser el adecuado para

impartirse en él, resulta en un aprendizaje eficaz para los alumnos y para un proceso de enseñanza efectivo para los profesores.

f. Propiciar altas expectativas en el estudiante. Los creadores de estos principios, proponen que al esperar más se obtendrá más. El establecer altas expectativas, influye en todos los tipos de estudiantes. Esto permite establecer ambientes de aprendizaje más amenos y conducentes para el aprendizaje de los estudiantes.
g. Respetar los diversos estilos de aprendizaje. Este principio significa que los estudiantes aprenderán de diversas formas. Esto quiere decir que algunos aprenderán más con la práctica y otros con la teoría. El respeto a estas formas de aprendizaje, podrá contribuir a un mejor desenvolvimiento del estudiante. Esto puede contribuir posteriormente a los estudiantes, para que aprendan nuevas formas de aprendizaje, las cuales no le son familiares.

1.2. Proceso enseñanza - aprendizaje

El entendimiento del proceso enseñanza – aprendizaje, radica en comprender los conceptos y la terminología básica sobre el mismo.

La enseñanza se fundamenta en una forma de transmitir información, utilizando la comunicación directa o empleando medios auxiliares (de mayor o menor grado de complejidad y costo). El proceso de enseñanza permite, que un profesor transmita sus conocimientos a uno o más alumnos, utilizando diversidad de medios, técnicas y/o herramientas de apoyo.

El aprendizaje es el proceso mediante el cual se adquiere un nuevo conocimiento, habilidad o capacidad. Esta adquisición debe poder repetirse en un futuro y poder resolver situaciones concretas, incluso diferentes a las que motivaron la adquisición del conocimiento, la habilidad o la capacidad. El aprendizaje ocurre algunas veces, como consecuencia de pruebas y errores, hasta lograr una solución válida. Otras veces puede darse por intuición, al resolver repentinamente un problema.

El proceso de enseñar permite al profesor mostrar o dar contenidos educativos, hacia los estudiantes, a través de diversos medios, en función de unos objetivos y dentro de un contexto. El proceso de aprender entonces complementa el proceso de enseñar. El estudiante intenta captar y retener el contenido que expone el profesor o cualquier otra fuente de información. Los objetivos que realiza el proceso de aprendizaje, pueden o no identificarse con los del profesor, pero se llevan a cabo dentro de un contexto.

El profesor antes de poder enseñar, debe ser consciente de los fines del proceso de enseñanza. Estos fines se agrupan en seis y son:

a. Conocimientos. Estos son aspectos de información, que se deben tener.
b. Comprensión. Esta es una capacidad, que permite el entendimiento de la información.
c. Aplicación. Esta es una capacidad de poder trasladar los planteamientos teóricos a situaciones reales.
d. Análisis. Esta capacidad permite poder descomponer un conjunto de información en sus partes o aspectos.
e. Síntesis. Esta capacidad permite componer en un todo o conjunto de información, los elementos y partes.

El profesor es un ente necesario en el proceso de enseñanza aprendizaje. Esto quiere decir que él debe promover un aprendizaje creativo, que logre estimular la comunicación y el espíritu del estudiante, lo que conlleva a las siguientes interacciones:

- Interacción profesor – conocimientos. Esta interacción permite indicar que el profesor es el diseñador de secuencias de aprendizaje, medios y materiales.
- Interacción conocimientos – alumno. Esta interacción permite indicar que el alumno debe gestionar de una manera crítica su aprendizaje.
- Interacción profesor – alumno. El profesor es un facilitador del aprendizaje, lo cual debe permitir al alumno aprender a aprender, por medio de un vínculo creado entre él y el profesor.

1.3. Métodos de clases magistrales

Una clase magistral es una forma de enseñanza, que permite transmitir ciertos conocimientos a un grupo de alumnos, que pasivamente escuchan y toman notas. Esto quiere decir que la mayor actividad se encuentra en las acciones que desarrolla el profesor (quien trata de que los alumnos comprendan el tema y que lo aprendan). El alumno se dedica a recibir la información y sus acciones son tomar nota del tema expuesto por el profesor.

La clase magistral puede poseer una variación. Esta variación radica en incitar al estudiante que forme parte de la presentación. El profesor es quien ahora presenta una clase más activa. Esta brinda al estudiante la posibilidad de

preguntar, exponer sus ideas con el consentimiento del profesor. La participación del estudiante permite interesarlo en el tema, pero se debe entender que siempre el profesor es quien maneja lo concerniente al desenvolvimiento del tema que expone.

El método permite que el profesor se instruya con anterioridad sobre el tema y luego lo presente a los estudiantes. La participación del estudiante permite aclarar dudas que les surjan en ese momento o que posteriormente les puedan surgir. Esto además permite que un alumno presente su punto de vista, sobre el tema expuesto y que se le retroalimente adecuadamente. El profesor puede de diversas formas verificar si el tema ha sido entendido o no por los estudiantes.

Las clases magistrales se pueden volver amenas si se utilizan los siguientes recursos:

- Imágenes visuales. Estas pueden ser el utilizar el pizarrón, diapositivas, películas o cualquier otra cosa, que permita de una forma visual captar la atención del estudiante para presentar el tema.
- Voz. El profesor debe contar con un volumen adecuado de voz. El profesor en caso de poseer una voz muy tenue, debe contar con un micrófono para poder modularla. La atención de los estudiantes puede perderse con un mal tono de voz.
- Cuerpo. La atención es más fácil de mantenerla si el expositor se encuentra de pie o moviéndose. Un expositor que se encuentra sentado, difícilmente podrá mantener la atención de los espectadores
- Plan de exposición. El brindar la distribución del plan de exposición al principio de la misma, permite al estudiante saber en todo momento en que parte de la exposición se encuentra y ubicarse conforme a los temas.

- Texto de la clase. La versión escrita de la clase promueve el aumento de la atención del estudiante. Esto beneficia su aprendizaje, debido a que puede ver, escuchar, leer, marcar o hacer anotaciones al texto de la clase magistral.
- Resumen final. Esta permite poder fijar por medio de las conclusiones de la exposición, los conocimientos para los estudiantes.

1.4. Aprendizaje práctico por medio de laboratorios

El aprendizaje práctico por medio de laboratorios se fundamenta en un paradigma, que se conoce como *learning by doing* (aprender haciendo). Este paradigma se caracteriza por exigir una alta interacción con el estudiante. El paradigma es casi desconocido, debido a que exige poseer factores tecnológicos y económicos altos. Esto quiere decir que exige poseer infraestructura adecuada, que muchas veces falta en centros educativos. El equivalente de éste paradigma, se constituye por medio de los laboratorios. Los estudiantes pueden poner en práctica los conocimientos que han adquirido, por medio de las clases magistrales.

El aprendizaje práctico exige que el profesor, sea considerado como socio responsable del proceso de aprendizaje del alumno. Esto contrasta con las clases magistrales, donde el alumno solamente es un receptor.

2. GUÍA DEL INSTRUCTOR

2.1. Definición de una guía del instructor

Una guía del instructor (*Instructor Guideline*), es un documento cuya finalidad es estructurar, estandarizar, planificar y optimizar todos los procesos del método de enseñanza – aprendizaje. Esto lo realiza por medio de una división en sesiones, las cuales poseen objetivos específicos y actividades específicas.

La división de sesiones del IG, permite añadir y mejorar el proceso enseñanza – aprendizaje. Las sesiones permiten incorporar una unidad temporal sobre los temas. Esto lo realiza planificando los contenidos y actividades para cada sesión, brindando el orden y la coordinación de las mismas.

El IG contiene de una forma escrita, la planificación que comprende el programa académico del curso. Un programa académico permite al profesor especificar actividades y estrategias que conlleven a un buen aprendizaje y que mejoren las actitudes de los alumnos.

El enfoque de un IG no es el de utilizarse una vez. Esto significa que ha sido diseñado de tal forma, que pueda servir para estandarizar un curso específico, aunque existan diversos catedráticos que impartan el mismo curso. El IG entonces se vuelve de uso exclusivo para los instructores, en este caso los catedráticos que imparten un curso determinado.

La gran adaptabilidad del IG permite podérsele añadir cambios, que sean necesarios debido al avance tecnológico que se presente. El incorporar algún tema o contenido al IG, precisa de un cuidadoso análisis y de una exhaustiva toma de consideraciones de principios para su aplicación eficaz.

Una guía del instructor así como la planificación del proceso de enseñanza – aprendizaje, cuentan con tres características fundamentales.

- Realista. El plan debe poder ser capaz de solventar las restricciones temporales, materiales, la capacidad de los estudiantes y las condiciones concretas del desarrollo de la enseñanza.
- Flexible. El plan debe ser capaz de adaptarse a nuevas circunstancias y poder prever diferentes alternativas.
- Preciso. El plan debe ser detallado. Las indicaciones deben ser exactas. Las actividades y objetivos claros y precisos.

2.2. Objetivos de una guía del instructor

La guía del instructor o *Instructor Guideline* es creado con el fin de estructurar, estandarizar, planificar y optimizar la enseñanza. Esto significa que un catedrático al emplearlo, minimiza la confusión con respecto a las políticas, que se toman durante un curso específico. El IG permite establecer claramente las expectativas y detalla el material, que se pretende que los alumnos aprendan.

2.3. Elementos y proceso de construcción de la guía del instructor

Los elementos importantes que integran un IG son:

- Datos generales del curso.
- Distribución del curso.
- Distribución del módulo.
- Evaluación.
- Detalle de sesiones.
- Distribución de sesiones.
- Detalle de conducción de tutoriales.
- Detalle de conducción de exámenes.
- Ejemplo de exámenes.
- Bibliografía.
- *Hand Book* (Libro de trabajo).

La construcción de un IG consiste en detallar cada uno de los elementos que lo componen. El mantener el orden de estos elementos, permitirá construir un IG propicio para poder realizar el proceso de enseñanza.

Los datos generales del curso son un elemento, que permite brindar una idea general de lo que es el curso. Este elemento se encuentra integrado por los siguientes apartados:

- Nombre del curso. El nombre viene indicado conforme el nombre que aparece en el pensum de la carrera. El código del curso se puede incluir también.

- Pre – requisitos. Esto se refiere a los cursos anteriores, que el curso actual necesita que sean aprobados antes de llevarlo.
- Post – requisitos. Esto se refiere a los cursos posteriores, que son abiertos al ganar el curso actual.
- Objetivo. El objetivo principal del curso.
- Módulos. Los diferentes nombres de los módulos en los cuales se integra el curso. Estos pueden ser un módulo o varios módulos.

La distribución del curso permite definir la forma en que se estructura el curso, conforme a la cantidad de horas y la cantidad de actividades que se realizan en él. Las horas son divididas en horas de teoría y horas de práctica, que poseerá el curso. Las actividades corresponden a exámenes, prácticas preparatorias y proyectos. La distribución se realiza utilizando una tabla. La tabla I muestra una distribución de horas y actividades, de tres módulos de ejemplo.

Tabla I. Distribución de horas y actividades de ejemplo

MÓDULO	TEORÍA	PRÁCTICA	TOTAL	EXAMEN	PRÁCTICA PREPARATORIA	PROYECTOS
MÓDULO - 1	4 horas	2 horas	6 horas	2	3	
MÓDULO - 2	2 horas	1 hora	3 horas	1	1	
MÓDULO - 3	4 horas	2 horas	6 horas	1	3	1
TOTAL	10 horas	5 horas	15 horas	4	7	1

La distribución del módulo permite dividir el mismo en sub módulos. Esto es debido a que dentro de un módulo, existirán conjuntos de contenido. La distribución presentará un listado de las aptitudes, que el estudiante obtendrá al finalizar el mismo. La tabla II muestra una distribución de un módulo de ejemplo.

Tabla II. Distribución del módulo de ejemplo

SUB-MÓDULO	TEORÍA	PRÁCTICA	TOTAL	AL FINALIZAR LOS SUB-MÓDULOS EL ESTUDIANTE OBTENDRÁ
SUB - MÓDULO 1	4 horas	0 horas	4 horas	Listado de aptitudes y capacidades del módulo.
SUB - MÓDULO 2	4 horas	2 horas	6 horas	Listado de aptitudes y capacidades del módulo.
TOTAL	8 horas	2 horas	10 horas	

La evaluación es el elemento que permite determinar el porcentaje, para de esta forma obtener el 100% de la nota del curso. La evaluación emplea una tabla que permite listar las actividades y los exámenes, para de esta forma llevar el control de las ponderaciones de las actividades. La tabla III muestra un detalle de evaluación de ejemplo.

Tabla III. Evaluación de ejemplo

ACTIVIDAD	NÚMERO	PORCENTAJE	TOTAL
PRÁCTICA PREPARATORIA	5	0.60 %	3.00 %
AUTO-EVALUACIÓN	2	1.00 %	2.00 %
LABORATORIO	1	40.0 %	40.00 %
EXAMEN PARCIAL	3	10.0 %	30.00 %
EXAMEN FINAL	1	25.0 %	25.00 %
		TOTAL	100.00 %

El detalle de sesiones se realiza por cada módulo que compone el curso. Este detalle incorpora el orden de las sesiones, así como la sesión en la cual se realizará una actividad (prácticas preparatorias o tareas). La tabla IV muestra un

detalle de sesiones de un módulo de ejemplo. Los exámenes se consideran como una sesión práctica. Las sesiones teóricas son todas aquellas donde el profesor brinda una clase magistral.

Tabla IV. Detalle de sesiones de ejemplo

NO	TEORÍA	NO	PRÁCTICA	PRÁCTICA PREPARATORIA / PROYECTO
1	SESION – 1			Práctica No. 1
2	SESION – 2			
3	SESION – 3			Proyecto No. 1
4	SESION – 4	5	SESION – 5	
6	SESION – 6			

La distribución de sesiones permite indicar la forma como se encuentra estructurada cada sesión. La distribución de sesiones se realiza por cada módulo, del que se encuentre formado el curso. La distribución debe tener en cuenta los conocimientos previos del alumno, lo que se espera obtener al finalizar la sesión, nivel de desarrollo intelectual del alumno, el entorno en el que se va a desarrollar la labor del docente y la duración de la sesión. Las sesiones se especifican con un número. Los módulos comprenden una serie de sesiones. La sesión debe obtener claramente el nombre del sub-módulo al que pertenece, el objetivo de la sesión, el listado de contenido propio de la sesión. Este contenido debe ser capaz de poderse establecer en un lapso temporal.

El detalle de conducción de tutoriales debe detallar el objetivo del tutorial, los criterios de evaluación del mismo, la documentación que contendrá, el módulo al que pertenece, el número de tarea, el objetivo de la tarea, la sesión en la que será asignada, en que sesión se entregará y cómo se calificará.

El detalle de conducción de exámenes permite establecer el criterio de calificación de los exámenes. El detalle corresponde a cada uno de los exámenes. El detalle debe mencionar el curso, el módulo, el total de exámenes

y el detalle de cada examen. El detalle de cada examen menciona el número de sesiones que abarca, la duración del examen y la forma subjetiva y objetiva.

El elemento de ejemplo de exámenes permite al profesor tomar una idea de la forma en que se debe evaluar y el criterio que debe poseer el examen.

La bibliografía comprende el conjunto de material didáctico. Este conjunto indica la referencia de donde se obtuvieron los temas para integrar cada sesión.

El *Hand Book* es considerado como un documento externo al IG. Este documento se encuentra fuera del mismo, debido a que comprende las actividades propias del curso. Este sería como un libro que contiene el listado de tareas, prácticas y proyectos. El Hand Book debe tener una descripción detallada de cada una de las actividades. El momento que se establece para que el catedrático deje las actividades, se encuentra establecido en el IG.

2.4. Modo de uso de la guía del instructor

El IG es para uso exclusivo del catedrático. El catedrático es tanto el ingeniero a cargo del curso, como el auxiliar del curso. El IG permite comprender fácilmente los contenidos y la forma en que se debe impartir el curso. Las sesiones establecen el contenido que debe cubrir el catedrático. Estas indican que el contenido no debe quedar interrumpido hasta terminar la sesión.

El IG establece que se utilice el Hand Book. El Hand Book permite a los catedráticos, el poder estandarizar sus actividades. El catedrático entonces es capaz de establecer una guía para las prácticas y proyectos.

3. GUÍA DEL INSTRUCTOR DE LA CLASE DE INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1

La guía del instructor del curso de Introducción a la Programación y Computación 1, se realizó conforme al formato sugerido en el curso de "Estructuración de cursos y laboratorios", impartido por el señor Mruntunjaya Panda.

3.1. Datos generales

NOMBRE DE CURSO:	**Introducción a la Programación y Computación 1 (770)**
PRE- REQUISITOS:	36 Créditos Matemática Básica 2 (103)
POST – REQUISITOS:	Introducción a la Programación y Computación 2 (771) Lenguajes Formales y de Programación (796)
OBJETIVO:	Al finalizar el curso, el estudiante poseerá la habilidad necesaria para poder realizar el diseño básico de programas que le

	permitirán resolver problemas mediante el uso de computadoras utilizando programación orientada a objetos. Plasmando los algoritmos que constituyen el programa en el lenguaje JAVA. Apoyándose en UML para poder diseñar los diagramas de clases de estos programas.
MÓDULOS:	**Programación principiantes**
VIGENCIA:	Primer Semestre 2009

3.2. Distribución del curso

El curso de Introducción a la Programación y Computación 1 se divide en veintiocho sesiones, integradas en un módulo denominado Programación principiantes. La tabla V muestra la tabla correspondiente a la distribución del curso de IPC1.

Tabla V. Distribución del curso de IPC1

MÓDULO	TEORÍA	PRÁCTICA	TOTAL	EXAMEN	PRÁCTICA PREPARATORIA	PROYECTOS
PROGRAMACIÓN PRINCIPIANTES	46horas	10 horas	56 horas	4	5	3
TOTAL	46horas	10 horas	56	4	5	3

horas

3.3. Distribución del módulo

El módulo de programación principiantes se divide en siete sub – módulos. La tabla VI muestra la tabla correspondiente a la distribución del módulo de IPC1.

Tabla VI. Distribución del módulo de IPC1

SUB-MÓDULO	TEORÍA	PRÁCTICA	TOTAL	AL FINALIZAR LOS SUB-MÓDULOS EL ESTUDIANTE OBTENDRÁ
INTRODUCCIÓN A LA COMPUTACIÓN Y PROGRAMACIÓN	8 horas	0 horas	8 horas	El conocimiento los conceptos básicos de computación. La comprensión sobre qué se considera *software*. El entendimiento de las diferencias entre los lenguajes de programación. La capacidad para resolver problemas computacionales. La comprensión sobre la consistencia de la programación estructurada y el entorno de programación.

PROGRAMACIÓN ORIENTADA A OBJETOS	14 horas	2 horas	14 horas	Los conocimientos necesarios para la comprensión de las distintas metodologías de programación. Los conceptos básicos y fundamentales de la programación orientada a objetos. El entendimiento de las ventajas que brinda UML al momento de ser aplicado en proyectos de *software*. La comprensión del concepto de polimorfismo al ser aplicado en conceptos específicos de la programación orientada a objetos. La comprensión de las ventajas de la herencia y la importancia que poseen los destructores. El entendimiento del concepto de polimorfismo aplicado a métodos virtuales. La habilidad de aplicar el concepto de construcciones abstractas.
INTRODUCCIÓN A LA PLATAFORMA JAVA	2 horas	0 horas	2 horas	El conocimiento sobre la evolución del lenguaje Java comprendiendo las características que hacen única a esta plataforma.
ELEMENTOS BÁSICOS DE PROGRAMACIÓN Y PROGRAMACIÓN ESTRUCTURADA	8 horas	2 horas	10 horas	La comprensión sobre los diferentes tipos de datos, constantes y variables utilizados en la programación estructurada. La habilidad para utilizar los operadores y generación de

				expresiones. El conocimiento necesario para identificar la importancia sobre: las estructuras de control, tanto iterativas como condicionales. La habilidad para realizar rutinas y aplicar la recursividad o recursión.
ESTRUCTURAS ALGORÍTMICAS	4 horas	0 horas	4 horas	La comprensión sobre las estructuras algorítmicas. La habilidad para utilizar arreglos y comprender los beneficios de utilizar cadenas dinámicas. La aplicación de los métodos de ordenamiento de datos.
FLUJOS (*STREAMS*) Y MANIPULACIÓN DE ARCHIVOS	4 horas	2 horas	6 horas	La habilidad para comprender y manipular archivos utilizando java. El entendimiento sobre el concepto de *streams*.
TIPO DE DATOS ABSTRACTOS	6 horas	4 horas	10 horas	La comprensión de la importancia del tipo de datos abstractos. La habilidad para utilizar pilas, colas y listas enlazadas utilizando java.
TOTAL	38 horas	18 horas	56 horas	

3.4. Evaluación

Ponderación de evaluaciones, tareas, prácticas y laboratorio. La tabla VII muestra la tabla correspondiente a la evaluación del curso de IPC1.

Tabla VII. Evaluación de IPC1

ACTIVIDAD	NÚMERO	PORCENTAJE	TOTAL
PRÁCTICA PREPARATORIA	5	0.60 %	3.00 %
AUTO-EVALUACIÓN	2	1.00 %	2.00 %
LABORATORIO	1	40.0 %	40.00 %
EXAMEN PARCIAL	3	10.0 %	30.00 %
EXAMEN FINAL	1	25.0 %	25.00 %
		TOTAL	**100.00 %**

3.5. Detalle de sesiones

La tabla VIII muestra la tabla correspondiente al detalle de sesiones del curso de IPC1.

Tabla VIII. Detalle de sesiones de IPC1

NO	TEORÍA	NO	PRÁCTICA	PRÁCTICA PREPARATORIA / PROYECTO
1	**PROGRAMACIÓN PRINCIPIANTES – 1**			
2	**PROGRAMACIÓN PRINCIPIANTES – 2**			
3	**PROGRAMACIÓN PRINCIPIANTES – 3**			
4	**PROGRAMACIÓN PRINCIPIANTES – 4**			
5	**PROGRAMACIÓN PRINCIPIANTES – 5**			
6	**PROGRAMACIÓN PRINCIPIANTES – 6**			
7	**PROGRAMACIÓN PRINCIPIANTES – 7**			
8	**PROGRAMACIÓN PRINCIPIANTES – 8**			
9	**PROGRAMACIÓN PRINCIPIANTES – 9**			
10	**PROGRAMACIÓN PRINCIPIANTES – 10**			Práctica Preparatoria No. 1
11	**PROGRAMACIÓN PRINCIPIANTES – 11**	12	**PROGRAMACIÓN PRINCIPIANTES – 12**	
13	**PROGRAMACIÓN PRINCIPIANTES – 13**			

14	**PROGRAMACIÓN PRINCIPIANTES – 14**			
15	**PROGRAMACIÓN PRINCIPIANTES – 15**			
16	**PROGRAMACIÓN PRINCIPIANTES – 16**			
17	**PROGRAMACIÓN PRINCIPIANTES – 17**	18	**PROGRAMACIÓN PRINCIPIANTES – 18**	Práctica Preparatoria No. 2
19	**PROGRAMACIÓN PRINCIPIANTES – 19**			
20	**PROGRAMACIÓN PRINCIPIANTES – 20**			Práctica Preparatoria No. 3
21	**PROGRAMACIÓN PRINCIPIANTES – 21**			
22	**PROGRAMACIÓN PRINCIPIANTES – 22**	23	**PROGRAMACIÓN PRINCIPIANTES – 23**	Práctica Preparatoria No. 4
24	**PROGRAMACIÓN PRINCIPIANTES – 24**			
25	**PROGRAMACIÓN PRINCIPIANTES – 25**	27	**PROGRAMACIÓN PRINCIPIANTES – 27**	Práctica Preparatoria No. 5
26	**PROGRAMACIÓN PRINCIPIANTES – 26**	28	**PROGRAMACIÓN PRINCIPIANTES – 28**	

3.6. Distribución de sesiones

La tabla IX muestra la tabla correspondiente a la distribución de sesiones del curso de IPC1.

Tabla IX. Distribución de sesiones de IPC1

SESIÓN NO. 1:	**SUB-MÓDULO: INTRODUCCIÓN A LA COMPUTACIÓN Y PROGRAMACIÓN**
OBJETIVO:	**Introducir a los estudiantes sobre los conceptos básicos de computación.**
CONTENIDO:	**1 Conceptos computacionales.** **1.1 Computadora.** **1.2 *Hardware*.** **1.3 *Software*.** **1.3.1 Programa.** **1.3.2 Información: instrucciones y datos.** **1.4 *Firmware.*** **2 Organización de la computadora.** **2.1 Diagrama de organización física.** **2.2 Dispositivos E/S.** **2.2.1 Dispositivos de entrada.** **2.2.2 Dispositivos de salida.** **2.2.3 Periféricos.** **2.3 CPU.** **2.3.1 Componentes primarios: ALU, CU, Registros.** **2.3.2 Palabra (*Word*).** **2.3.3 Microprocesadores y velocidad.** **2.4 Memoria intermedia (caché).** **2.5 Memoria principal.** **2.5.1 Unidad elemental de memoria: *Bits, Bytes*.** **2.5.2 Sistema binario** **2.5.3 Tabla de unidades de medida de almacenamiento.** **2.5.4 Tipos de memoria.** **2.6 Memoria secundaria.** **2.6.1 Archivos.** **2.6.2 Tipos de dispositivos: magnéticos, ópticos, etc.**

	2.7 Comparación entre memorias: almacenamiento, velocidad y precio.
SESIÓN NO. 2:	**SUB-MÓDULO: INTRODUCCIÓN A LA COMPUTACIÓN Y PROGRAMACIÓN**
OBJETIVO:	**Presentar a los estudiantes en qué consiste el software y los lenguajes de programación.**
CONTENIDO:	**1 El *software*.** **1.1 Programa.** **1.2 División del *software*.** **1.2.1 *Software* del sistema.** **1.2.1.1 Sistema operativo: definición y características.** **1.2.1.2 Compiladores e intérpretes.** **1.2.1.3 Programas de utilidad.** **1.2.2 *Software* de aplicación.** **1.2.3 Relación entre software del sistema y software de aplicación.** **2 Lenguajes de programación.** **2.1 Lenguajes de máquina.** **2.1.1 Características.** **2.1.2 Conjuntos de instrucciones: CISC y RISC.** **2.1.3 Desventajas.** **2.2 Lenguajes de bajo nivel.** **2.2.1 Características.** **2.2.2 Lenguaje ensamblador: instrucciones nemotécnicas.** **2.3 Lenguajes de alto nivel.** **2.3.1 Características.** **2.3.2 Ventajas.** **2.3.3 Desventajas.**
SESIÓN NO. 3:	**SUB-MÓDULO: INTRODUCCIÓN A LA COMPUTACIÓN Y PROGRAMACIÓN**

OBJETIVO:	**Brindar a los estudiantes la capacidad para resolver problemas computacionales.**
CONTENIDO:	**1 Algoritmo.** **1.1 Concepto e importancia: origen** **1.2 Características: preciso, definido, finito.** **1.3 Diferencia entre métodos algorítmicos y heurísticos.** **2 Resolución de problemas computacionales.** **2.1 Análisis del problema.** **2.1.1 Concepto e importancia.** **2.1.2 Definición de alcances e identificación de limitantes.** **2.2 Diseño del algoritmo.** **2.2.1 Concepto e importancia.** **2.2.2 Introducción al diseño descendente (*top-down*) o modular.** **2.2.2.1 Programación modular.** **2.2.2.2 Módulo: módulo principal, sub-módulos.** **2.2.2.3 Características y beneficios.** **2.2.3 Herramientas de diseño.** **2.2.3.1 Diagramas de flujo.** **2.2.3.2 Pseudocódigo.** **2.2.3.3 Ejemplo de diseño.** **2.3 Codificación.** **2.3.1 Características básicas y mejores prácticas.** **2.4 Compilación y ejecución.** **2.4.1 Fases de la compilación.** **2.4.2 Programa fuente, objeto y ejecutable.** **2.5 Verificación y depuración.** **2.5.1 Verificación.** **2.5.2 Depuración.** **2.5.3 Tipos de errores: de compilación, de ejecución y lógicos.** **2.6 Documentación.** **2.6.1 Documentación interna.** **2.6.2 Documentación externa.** **2.7 Mantenimiento.** **2.7.1 Importancia como factor de éxito.**

	2.7.2 Versionamiento: <mayor>.<menor>.<control>.<*build*>.
SESIÓN NO. 4:	**SUB-MÓDULO: INTRODUCCIÓN A LA COMPUTACIÓN Y PROGRAMACIÓN**
OBJETIVO:	**Introducir a los estudiantes el concepto de programación estructurada y el entorno de programación.**
CONTENIDO:	**1 Programación estructurada.** **1.1 Definición, reglas y beneficios.** **1.2 Recursos abstractos.** **1.3 Diseño descendente** **1.3.1. Definición** **1.3.2. Perspectivas** **1.3.3. Refinamiento Sucesivo** **1.4 Estructuras de control** **1.4.1. Definición** **1.4.2. Secuencia** **1.4.3. Selección** **1.4.4. Repetición** **1.5 Teorema de la programación estructurada** **1.5.1. Estructuras básicas.** **1.5.2. Reglas** **2 Entorno de programación.** **2.1 Definición y componentes.** **2.2 Entorno integrado de desarrollo.**
SESIÓN NO. 5:	**SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS**
OBJETIVO:	**Presentar a los estudiantes las diferentes metodologías de programación.**

CONTENIDO:	**1 Metodologías de programación.** **1.1 Definición.** **1.2 Programación procedural (*top-down*).** **1.3 Programación orientada a objetos.** **1.4 Programación orientada a eventos.** **1.5 Programación orientada a servicios.** **2 Concepto de abstracción y clasificación.**
SESIÓN NO. 6:	**SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS**
OBJETIVO:	**Introducir a los estudiantes a la programación orientada a objetos.**
CONTENIDO:	**1 Conceptos básicos: definiciones y características.** **1.1 Clase: tipo de objeto.** **1.2 Objeto: instancia, propiedades y operaciones.** **1.3 Mensaje.** **1.4 Método: enlace o ligadura tardío (dinámico) y temprano (estático).** **2 Principio de encapsulamiento y ocultación de información: principios de Parnas.** **3 Los miembros de una clase.** **3.1 Atributos (propiedades).** **3.2 Métodos (definición del comportamiento: operaciones).** **4 Modificadores básicos de visibilidad.** **4.1 Público.** **4.2 Privado.** **4.3 Razones de encapsular** **5 Relaciones.** **5.1 A nivel de clases: herencia.** **5.2 A nivel de objetos: agregación/composición y asociación.**

SESIÓN NO. 7:	**SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS**
OBJETIVO:	**Presentar las ventajas que permite el lenguaje unificado de modelado al momento de aplicarlo en los proyectos de *software*.**
CONTENIDO:	**1 Introducción a UML: Lenguaje unificado de modelación.** **1.1 Modelación visual: definición e importancia.** **1.2 UML: definición, evolución y clasificación de diagramas.** **1.3 Diagrama estático de clases: principios básicos.** **1.3.1 Definición y características.** **1.3.2 Artefactos gráficos principales: clases y sus componentes.** **1.3.3 Atributos: modificadores de visibilidad, valores y restricciones.** **1.3.4 Relación de herencia.** **1.3.5 Relación de agregación/composición y asociación.** **1.3.6 Multiplicidad y dirección de las relaciones a nivel de objetos.** **1.3.7 Roles y nombramiento de asociaciones.** **1.3.8 Relaciones múltiples: clases asociativas y relaciones reflexivas.** **1.3.9 Dependencia.**
SESIÓN NO. 8:	**SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS**
OBJETIVO:	**Entender en qué consiste el polimorfismo y conceptos específicos de la programación orientada a objetos.**
CONTENIDO:	**1 Polimorfismo.** **1.1 Definición y características.** **1.2 Polimorfismo ad hoc: sobrecarga de métodos.** **1.3 Polimorfismo puro.**

	2 Conceptos avanzados. **2.1 Prototipo de la definición de métodos: firma/signatura y sobrecarga.** **2.2 Visibilidad y modificadores de acceso: acceso semiprivado (protegido).** **2.3 Miembros de instancia: variables de ejemplares.** **2.4 Miembros estáticos (*static*): datos y métodos a nivel de clase.** **2.5 Definición de clase** **2.6 Creación de instancia con *new*.** **2.7 Contexto de ejecución: clase estática y clase dinámica.** **2.8 Constructores: reglas, sobrecarga, inicializadores en herencia.**
SESIÓN NO. 9:	**SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS**
OBJETIVO:	**Aprender la importancia de los destructores y de las ventajas que permite la herencia.**
CONTENIDO:	**1 Conceptos Avanzados** **1.1 Destructor.** **1.1.1 Modos de programación: real y protegido.** **1.1.2 Recolector de basura (GC): administración de memoria.** **1.1.3 Modelo de control de objetos: *Stack, Memory-Heap*.** **1.1.4 Destrucción de objetos y ejecución del destructor.** **1.1.5 Método *finalize*().** **1.2 Referencia "*this*".** **1.3 Clases parametrizadas: plantilla de clases o *generics*.** **2 Herencia.** **2.1 Concepto como expansión y contracción.** **2.2 Beneficios y costos.** **2.3 Heurísticas de diseño: combinación (herencia múltiple).**
SESIÓN NO. 10:	**SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS**

OBJETIVO:	**Aprender sobre el polimorfismo aplicado a métodos virtuales.**
CONTENIDO:	**1 Polimorfismo y métodos virtuales (virtualización).** **1.1 Sobrecarga de métodos: coerción y signatura de argumentos.** **1.2 Sobrecarga de operadores.** **1.3 Anulación: refinamiento y reemplazo.** **1.4 final <método> (evitar sobrecarga).** **1.5 Métodos virtuales.** **1.5.1. Métodos virtuales puros, diferidos o genéricos.**
SESIÓN NO. 11:	**SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS**
OBJETIVO:	**Introducir a los estudiantes el concepto de construcciones abstractas.**
CONTENIDO:	**1 Construcciones abstractas.** **1.1 Clase abstracta o diferida.** **1.2 Métodos abstractos.** **1.3 *abstract*, enlace tardío (dinámico) y polimorfismo.** **1.4 Interface.** **1.4.1 Diferencia entre herencia e implementación de interface.** **1.4.2 Herencia (simple), interfaces (múltiples).** **1.4.3 Jerarquía de interfaces.** **1.4.4 Variables de interfaces.** **1.5 Clases concretas, clases abstractas e interfaces.** **2 Conceptos adicionales.** **2.1 Relación entre clase y tipo de dato.** **2.1.1 Tipos primitivos o fundamentales.** **2.1.2 Tipos derivados.** **2.1.3 Tipos elaborados de usuarios.** **2.2 Enfoque de diseño modular: alta cohesión y poco acoplamiento.**

	2.3 Dominio.
SESIÓN NO. 12:	**SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS**
OBJETIVO:	**Evaluar los contenidos de las sesiones 1- 11 aprendidos por los estudiantes.**
CONTENIDO:	**Examen Parcial número 1.**
SESIÓN NO. 13:	**SUB-MÓDULO: INTRODUCCIÓN A LA PLATAFORMA JAVA**
OBJETIVO:	**Explicar al estudiante la evolución del lenguaje java, presentándole características de esta plataforma.**
CONTENIDO:	**1 Evolución del lenguaje Java.** **1.1 Lenguaje B.** **1.2 Lenguaje C: UNIX, K&R, ANSI C, ventajas.** **1.3 Lenguaje C++ (C con clases): características, ANSI/ISO C++.** **1.4 Lenguaje Java: proyectos Green, *Oak* (*Sun Microsystems*) y la WWW.** **2 Definición y características.** **2.1 Arquitectura neutra: Java *Virtual Machine* (JVM).** **2.2 Proceso de ejecución: *bytecode*, intérprete, *Just-In-Time*.** **2.3 Portabilidad, estandarización, robustez, multihilo, seguro, POO.** **3 Plataforma Java.** **3.1 Tipos de programas: aplicaciones y *applets*.** **3.2 JDK y versiones de la plataforma: J2EE, J2SE, J2ME.** **3.3 JCP.** **4 Aspectos de compilación y ejecución de programas Java.**

	5 Estructura general de un programa de aplicación en Java.
SESIÓN NO. 14:	**SUB-MÓDULO: ELEMENTOS BÁSICOS DE PROGRAMACIÓN Y PROGRAMACIÓN ESTRUCTURADA**
OBJETIVO:	**Aprender sobre los diferentes tipos de datos, constantes y variables utilizados en la programación estructurada.**
CONTENIDO:	**1 Tipos de datos.** **1.1 Primitivos.** **1.1.1 Enteros (*int, byte, short, long*): conversiones, base-x (10|16|8).** **1.1.2 Números de coma flotante (*float, double*).** **1.1.3 Caracteres (*char*): codificación (ASCII, *Unicode*), tratamiento.** **1.1.4 Lógicos (*boolean*).** **1.2 Referenciados.** **1.3 *void*.** **2 Constantes: literales y declaradas (final).** **3 Variables.** **3.1 Concepto y manipulación.** **3.2 Declaración y definición (asignación).** **3.3 Inicialización.** **3.4 Notación estándar: estilo y legibilidad.** **4 Entorno, ámbito o alcance de variables.** **4.1 Variables locales (ámbito de método): automáticas.** **4.2 Variables de módulo: ámbito de la clase.** **5 Entradas y salidas de datos**
SESIÓN NO. 15:	**SUB-MÓDULO: ELEMENTOS BÁSICOS DE PROGRAMACIÓN Y PROGRAMACIÓN ESTRUCTURADA**

OBJETIVO:	**Instruir a los estudiantes sobre el uso de los operadores y la generación de expresiones y presentar la importancia de las estructuras de control.**
CONTENIDO:	**1 Expresiones y Operadores.** **1.1 Expresión.** **1.2 Operadores y clasificación por operandos: unarios, binarios, ternarios.** **1.3 Operador de asignación.** **1.4 Operadores aritméticos: precedencia, asociatividad, evaluación y uso de paréntesis.** **1.5 Operadores de incrementación y decrementación: notación prefija y postfija.** **1.6 Operadores relacionales: precedencia y evaluación.** **1.7 Operadores lógicos o booleanos.** **1.7.1 Tablas de verdad: negación, O-exclusivo, Y-lógico, O-lógico.** **1.7.2 Evaluación en cortocircuito.** **1.8 Operadores de manipulación de bits: lógicos, complemento, desplazamientos.** **1.9 Otros operadores: condicional (?:), coma (,), punto (.), (), [], *new*, *instanceof*.** **2 Conversiones de tipos: implícitas y explícitas.** **3 Estructuras de control.** **3.1 Concepto e importancia.** **3.2 Tipos: secuencia, selección (condición), repetición.** **3.3 Sentencia compuesta.**
SESIÓN NO. 16:	**SUB-MÓDULO: ELEMENTOS BÁSICOS DE PROGRAMACIÓN Y PROGRAMACIÓN ESTRUCTURADA**
OBJETIVO:	**Introducir a los estudiantes a los conceptos de estructuras de control condicionales e iterativas.**

CONTENIDO:	**1 Estructuras de control condicionales.** **1.1 Si – Sino (*if – else*): partes, diagrama de flujo, sentencias anidadas.** **1.2 En caso (*switch / case*): partes, evaluación.** **1.3 Consideraciones adicionales: expresión condicional, evaluación en cortocircuito, estilo y diseño, errores.** **2 Estructuras de control repetitivas o iterativas (bucles, *loops*).** **2.1 Conceptos básicos.** **2.1.1 Bucle, cuerpo, iteración, consideraciones de diseño.** **2.1.2 Variable de control, etapas (inicialización, prueba, actualización).** **2.2 Mientras (*while*): partes, diagrama de flujo.** **2.3 Para (*for*): partes, diagrama de flujo, sentencias nulas y bucles vacíos.** **2.4 Repetir – Hasta / Hacer - Mientras (*Repeat – Until / do-while*): partes, diagrama de flujo, diferencias con ciclo Mientras.** **2.5 Comparación entre los diferentes bucles: formatos y recomendaciones.** **2.6 Anidación de bucles: externos e internos.** **2.7 Diseño eficiente de bucles: patrones estructurales.** **2.7.1 Cero iteraciones y bucles infinitos.** **2.7.2 Manejo de incrementos y decrementos.** **2.7.3 Terminaciones anormales de un ciclo.** **2.7.4 Modelos: control por centinelas y por indicadores (banderas).** **2.7.5 Precauciones con el manejo de las variables de control.** **2.7.6 Bucles para diseño de sumas y productos.** **2.7.7 Fin de un bucle: por tamaño, por pregunta, valor centinela, etc.**
SESIÓN NO. 17:	**SUB-MÓDULO: ELEMENTOS BÁSICOS DE PROGRAMACIÓN Y PROGRAMACIÓN ESTRUCTURADA**
OBJETIVO:	**Introducir a los estudiantes los conceptos de rutinas y recursividad o recursión.**
CONTENIDO:	**1 Las rutinas (métodos).** **1.1 Concepto, bloques de construcción y partes (cabecera y cuerpo).** **1.2 Procedimiento y función.** **1.3 Método *main*(), línea de comandos.**

	1.4 El valor de retorno: devolución de valores (sentencia *return*). **1.5 Llamadas (invocaciones) a métodos.** **1.6 [Corolario] Modificadores de visibilidad (acceso).** **1.7 [Corolario] Entorno de las variables (ámbito o alcance).** **1.7.1 Ámbito de la clase.** **1.7.2 Ámbito de método y de bloque: variables locales.** **1.8 Los parámetros.** **1.8.1 Paso de argumentos por valor (paso por copia).** **1.8.2 Paso de argumentos por referencia (paso por variable).** **1.8.3 Lista de parámetros múltiples.** **1.8.4 Parámetro final.** **2 Recursividad o recursión.** **2.1 Concepto, naturaleza/comportamiento e importancia.** **2.2 Recursividad directa.** **2.3 Recursividad indirecta: métodos mutuamente recursivos.** **2.4 Recursión vrs. Iteración: ventajas y desventajas, directrices de diseño.** **2.5 Diseño: condición de terminación, recursión infinita.** **2.6 Ejemplos y aplicación en modelo matemáticos.**
SESIÓN NO. 18:	**SUB-MÓDULO: ELEMENTOS BÁSICOS DE PROGRAMACIÓN Y PROGRAMACIÓN ESTRUCTURADA**
OBJETIVO:	**Evaluar los contenidos de las sesiones 13- 17 aprendidos por los estudiantes.**
CONTENIDO:	**Examen Parcial número 2**
SESIÓN NO. 19:	**SUB-MÓDULO: ESTRUCTURAS ALGORÍTMICAS**
OBJETIVO:	**Lograr que el estudiante entienda en qué consisten los arreglos y los beneficios de utilizar cadenas dinámicas.**

CONTENIDO:	**1 Arreglos de datos.** **1.1 Conceptos: elementos, longitud, indexación, representación en memoria.** **1.2 Declaración, subíndices, atributo *length*, inicialización, copia de arreglos.** **1.3 Arreglos bidimensionales (matrices).** **1.3.1 Declaración, inicialización.** **1.3.2 Representación en memoria.** **1.3.3 Acceso a elementos: uso de bucles.** **1.4 Arreglos n-dimensionales (multidimensionales): aplicación práctica.** **1.5 Manejo de arreglos como parámetros.** **2 Cadenas de caracteres.** **2.1 Concepto: manejo en memoria, diferencia con arreglos de caracteres.** **2.2 Cadenas estáticas (clase *String*).** **2.2.1 Declaración, inicialización/constructores, asignación, ejemplos.** **2.2.2 Manejo como parámetros y arreglo de cadenas.** **2.2.3 Principales operaciones y métodos: longitud, concatenación, obtención de caracteres, comparación de cadenas, conversión de cadenas, búsqueda de caracteres y sub-cadenas.** **2.3 Cadenas dinámicas (clase *StringBuffer*): métodos propios para añadir, insertar, eliminar y reemplazar.** **2.4 Cadenas especializadas: clase *StringTokenizer*.**
SESIÓN NO. 20:	**SUB-MÓDULO: ESTRUCTURAS ALGORÍTMICAS**
OBJETIVO:	**Aprender sobre los métodos de ordenamientos de datos y cuando es mejor utilizar cada uno de ellos.**
CONTENIDO:	**1 Ordenamiento de datos en arreglos.** **1.1 Concepto: algoritmos y análisis de rendimiento (notación-O).** **1.2 Por intercambio (*swap*).** **1.3 Por selección.**

	1.4 Por inserción. **1.5 Burbuja (*bubblesort*).** **1.6 *QuickSort*.** **2 Búsqueda de datos en arreglos.** **2.1 Secuencial: concepto, algoritmo, análisis de rendimiento.** **2.2 Binaria: concepto, algoritmo, análisis de rendimiento.**
SESIÓN NO. 21:	**SUB-MÓDULO: FLUJOS (STREAMS) Y MANIPULACIÓN DE ARCHIVOS**
OBJETIVO:	**Introducir a los estudiantes el concepto de streams y su aplicación.**
CONTENIDO:	**1 Concepto de flujo: modelo productor-consumidor, canal de conexión (*pipe*).** **2 Tipos de flujos.** **2.1 Jerarquía de clases: Java y C#.** **2.2 Definición y operaciones p/archivos.** **2.3 Clases Filtro: definición, jerarquía, operaciones.** **3 Concepto de archivo.** **3.1 Persistencia de datos.** **3.2 Características estructurales: formato.** **3.3 Representación, manipulación.** **3.4 Clase Archivo.**
SESIÓN NO. 22:	**SUB-MÓDULO: FLUJOS (STREAMS) Y MANIPULACIÓN DE ARCHIVOS**
OBJETIVO:	**Aprender sobre la manipulación de archivos.**
	1 Tipos de archivos: texto y binarios. **2 Estructura de archivos.**

CONTENIDO:	**2.1 Registros (colección lógica de datos): concepto y definición por campos.** **2.2 Manipulación de registros.** **2.3 Tamaño de registros y cálculo para acceso lógico por bloques/campos.** **3 Operaciones básicas con archivos.** **3.1 Abrir y cerrar.** **3.2 Lectura, escritura y posicionamiento.** **3.3 Localización del final del archivo.** **4 Clasificación fundamental de archivos según su organización.** **4.1 Organización secuencial: concepto, manipulación (lectura, escritura).** **4.2 Organización directa o aleatoria (acceso aleatorio).** **4.2.1 Concepto, manipulación.** **4.2.2 Clase Archivo Acceso Aleatorio: creación de objeto, métodos de posicionamiento, proceso general para agregar y consultar registros.**
SESIÓN NO. 23:	**SUB-MÓDULO: FLUJOS (STREAMS) Y MANIPULACIÓN DE ARCHIVOS**
OBJETIVO:	**Evaluar los contenidos de las sesiones 19- 22 aprendidos por los estudiantes.**
CONTENIDO:	**Examen Parcial número 3.**
SESIÓN NO. 24:	**SUB-MÓDULO: TIPO DE DATOS ABSTRACTOS**
OBJETIVO:	**Aprender sobre la utilización y el manejo de las pilas.**
CONTENIDO:	**1 Fundamentos teóricos.** **1.1 Memoria dinámica y apuntadores: concepto de *null*.** **1.2 Introducción a la clasificación de estructuras: unidimensionales (pilas, colas y listas), bidimensionales (árboles: binario, AVL), n-dimensionales**

	(árboles balanceados B), matrices esparcidas (listas ortogonales), tablas de Hash. **2 Pilas (*Stack*).** **2.1 Concepto: política de acceso a datos (LIFO), estructura e importancia.** **2.2 Especificación de operaciones.** **2.2.1 *push, pop, top, reset*.** **2.2.2 Pila vacía (desbordamiento negativo, *underflow*).** **2.2.3 Pila llena (desbordamiento positivo, *overflow*).** **2.2.4 Tamaño de la pila.** **2.3 Implementación estática utilizando arreglos: definición de la clase pila.** **2.4 Implementación dinámica: clase Pila.**
SESIÓN NO. 25:	**SUB-MÓDULO: TIPO DE DATOS ABSTRACTOS**
OBJETIVO:	**Aprender sobre la utilización y manejo de las colas.**
CONTENIDO:	**1 Colas (*Queues*).** **1.1 Introducción a teoría de colas.** **1.2 Concepto: política de acceso a datos (FIFO), estructura e importancia.** **1.3 Especificación de operaciones.** **1.3.1 *push, pop, top, reset*.** **1.3.2 Cola vacía, cola llena, tamaño de la cola.** **1.4 Implementación estática utilizando arreglos.** **1.4.1 Modelos de manipulación: simple y circular.** **1.4.2 Definición de la clase cola.** **1.5 Implementación dinámica: clase Cola.** **1.6 Aplicación: cola de prioridades (modelo y definición).** **2 Listas enlazadas.** **2.1 Nodo: concepto y estructura interna.** **2.2 Definición y representación en memoria.** **2.3 Clasificación:** **2.3.1 Simplemente enlazadas.** **2.3.2 Doblemente enlazadas.** **2.3.3 Circular simplemente enlazada.**

	2.3.4 Circular doblemente enlazada.
SESIÓN NO. 26:	**SUB-MÓDULO: TIPO DE DATOS ABSTRACTOS**
OBJETIVO:	**Aprender sobre la utilización y manejo de las listas.**
CONTENIDO:	**1. Listas enlazadas** **1.1 Estructura y operaciones:** **1.1.1 Definición de nodo.** **1.1.2 Propiedades o atributos: referencias de cabecera y cola.** **1.1.3 Creación e inicialización.** **1.1.4 Inserción: en la cabeza, en el final, antes de y después de.** **1.1.5 Búsqueda.** **1.1.6 Eliminación.** **1.1.7 Recorrido y comprobación de vacío.** **2 Listas doblemente enlazadas: concepto, representación, nodo binario, características, implementación de operaciones.** **3 Listas circulares: concepto, representación, implementación de operaciones.** **4 Listas enlazadas genéricas: clase Objeto, clase ListaEnlazada**
SESIÓN NO. 27:	**SUB-MÓDULO: TIPO DE DATOS ABSTRACTOS**
OBJETIVO:	**Permitir a los estudiantes la resolución de dudas que hayan generado en el transcurso de las sesiones.**
CONTENIDO:	**Repaso de las sesiones 1 - 27.**

SESIÓN NO. 28:	**SUB-MÓDULO: TIPO DE DATOS ABSTRACTOS**
OBJETIVO:	**Evaluar los contenidos de las sesiones 1- 27 aprendidos por los estudiantes.**
CONTENIDO:	**Examen Final.**

3.7. Detalle de conducción de tutoriales

La tabla X muestra la tabla correspondiente al detalle de conducción de tutoriales del curso de IPC1.

Tabla X. Detalle de conducción de tutoriales de IPC1

ACTIVIDAD	DETALLE
Iniciado en sesión de clase número:	N.A.
Finalizado en sesión de clase número	N.A.
Viva / *Demo* para ser conducido:	N.A.
Criterio de evaluación:	N.A.
Viva:	N.A.

Documentación:	N.A.

3.8. Detalle de conducción de exámenes

La tabla XI muestra la tabla correspondiente al detalle de conducción de exámenes del curso de IPC1.

Tabla XI. Detalle de conducción de exámenes de IPC1

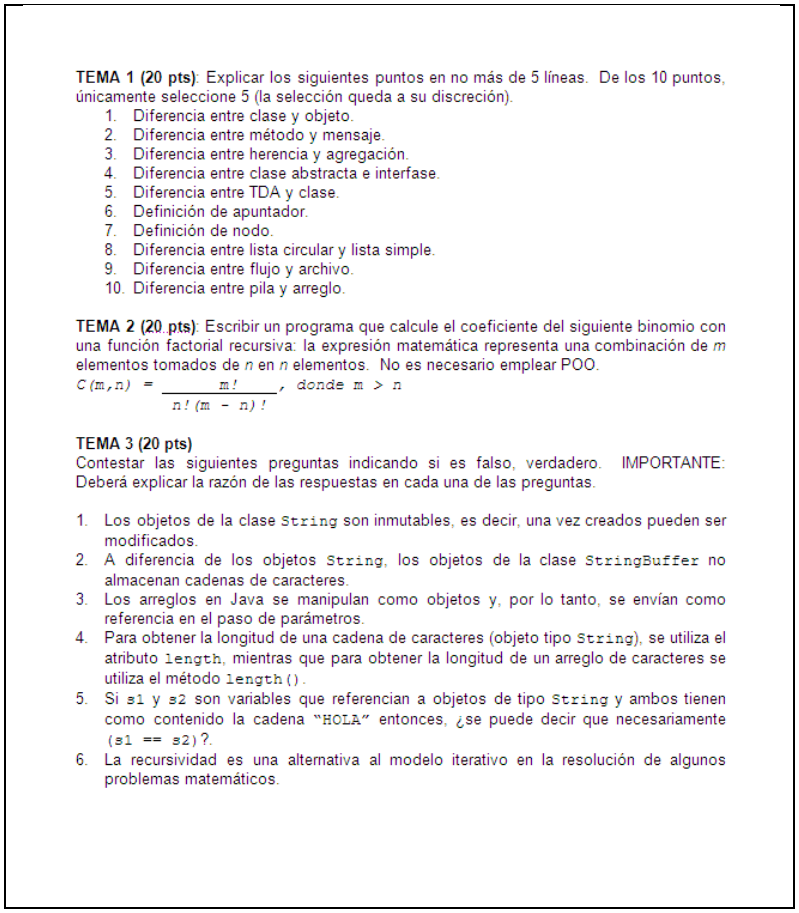
EXAMEN NÚMERO:	**1**
Examen a ser realizado después de la sesión número:	12
Temas para el examen:	Sesión 1 – Sesión 11
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%
EXAMEN NÚMERO:	**2**
Examen a ser realizado después de la sesión número:	17
Temas para el examen:	Sesión 13 – Sesión 17
Duración del Examen:	2 horas

Distribución del Examen:	Objetivo – 100%
EXAMEN NÚMERO:	**3**
Examen a ser realizado después de la sesión número:	22
Temas para el examen:	Sesión 18 – Sesión 22
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%
EXAMEN NÚMERO:	**4**
Examen a ser realizado después de la sesión número:	27
Temas para el examen:	Sesión 1 – 27, haciendo especial énfasis en las sesiones 24 – 27
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%

3.9. Ejemplo del examen

La figura 1 muestra el Ejemplo de examen curso de IPC1, correspondiente al examen final.

Figura 1. Ejemplo de examen final de IPC1



TEMA 1 (20 pts): Explicar los siguientes puntos en no más de 5 líneas. De los 10 puntos, únicamente seleccione 5 (la selección queda a su discreción).

1. Diferencia entre clase y objeto.
2. Diferencia entre método y mensaje.
3. Diferencia entre herencia y agregación.
4. Diferencia entre clase abstracta e interfase.
5. Diferencia entre TDA y clase.
6. Definición de apuntador.
7. Definición de nodo.
8. Diferencia entre lista circular y lista simple.
9. Diferencia entre flujo y archivo.
10. Diferencia entre pila y arreglo.

TEMA 2 (20 pts): Escribir un programa que calcule el coeficiente del siguiente binomio con una función factorial recursiva: la expresión matemática representa una combinación de *m* elementos tomados de *n* en *n* elementos. No es necesario emplear POO.

$$C(m,n) = \frac{m!}{n!(m-n)!}, \text{ donde } m > n$$

TEMA 3 (20 pts)

Contestar las siguientes preguntas indicando si es falso, verdadero. IMPORTANTE: Deberá explicar la razón de las respuestas en cada una de las preguntas.

1. Los objetos de la clase `String` son inmutables, es decir, una vez creados pueden ser modificados.
2. A diferencia de los objetos `String`, los objetos de la clase `StringBuffer` no almacenan cadenas de caracteres.
3. Los arreglos en Java se manipulan como objetos y, por lo tanto, se envían como referencia en el paso de parámetros.
4. Para obtener la longitud de una cadena de caracteres (objeto tipo `String`), se utiliza el atributo `length`, mientras que para obtener la longitud de un arreglo de caracteres se utiliza el método `length()`.
5. Si `s1` y `s2` son variables que referencian a objetos de tipo `String` y ambos tienen como contenido la cadena `"HOLA"` entonces, ¿se puede decir que necesariamente `(s1 == s2)`?.
6. La recursividad es una alternativa al modelo iterativo en la resolución de algunos problemas matemáticos.

7. En Java, las variables de los tipos de datos primitivos o básicos, al momento de declararlas quedan definidas.
8. Variables automáticas, por defecto, son aquellas que se declaran localmente en un método.
9. Por lo general, las variables de control de los bucles deben ser: (1) Inicializadas; (2) Comprobadas o evaluadas; (3) Iteradas.
10. La sentencia `continue` produce una salida inmediata de un bloque, mientras que la sentencia `break` realiza una finalización anormal de un bloque.

TEMA 4 (20 pts): Dadas las siguientes entidades, construir un modelo UML de clases: el diseño consiste en implementar una jerarquía de tipos de datos numéricos que extienda los tipos de datos fundamentales tales como *long* y *double*. Así, las entidades son:

- **Real**: conjunto de números racionales e irracionales.
- **Entero**: puede ser positivo y negativo.
- **Natural**: conjunto de números enteros definidos en el siguiente rango `[0,1,2,…,+∞]`.
- **Racional**: se puede representar como fracción (numerador/denominador).
- **Irracional**: no se puede representar como fracción → `e, π, √2, √5`.
- **Imaginario**: conjunto de números imaginarios → `Real * i,` en donde `i = √-1`.
- **Complejo**: se define como un número Real + un número Imaginario.
- **Vector**: arreglo unidimensional de números complejos.
- **Matriz**: arreglo bidimensional de números complejos.

Observaciones:

- Indicar los atributos/propiedades de las clases: para el efecto, utilice sólo datos de tipo *long* y/o *double*.
- No es necesario indicar las operaciones de todas las clases; sin embargo, sí se recomienda dejar indicadas algunas (las que Ud. considere)
- Es ESENCIAL indicar las relaciones entre clases (si aplica): herencia, asociación y agregación.

TEMA 5 (20 pts): Escriba una breve función en Java, potencia_de_dos, que tome un int i y retorne true si y sólo si i es una potencia de 2. Sin embargo, esa función no puede usar multiplicación ni división ni los métodos de potenciación/radicación de la clase Math.

3.10. Bibliografía recomendada

La tabla XII muestra la tabla correspondiente a la bibliografía recomendada para el curso de IPC1.

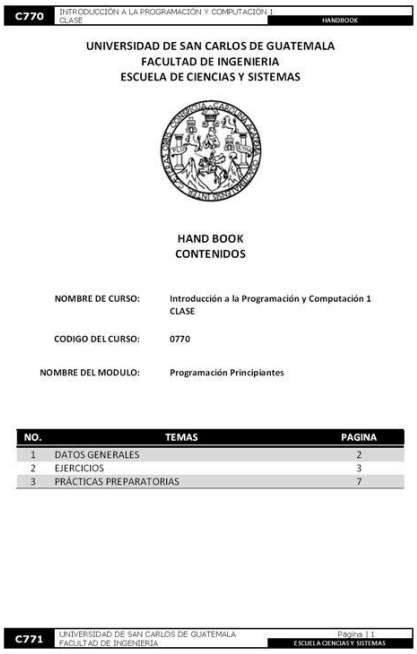
Tabla XII. Bibliografía recomendada de IPC1

No.	Libro
1	Joyanes, L. y Zahonero, I. **Programación en Java 2 (algoritmos, estructura de datos y programación orientada a objetos)**. Editorial *McGraw-Hill* / Interamericana de España, S. A. 2ª Edición. España. Año 2002.
2	Budd, Timothy. **Introducción a la programación orientada a objetos**. Editorial *Addison-Wesley*. Iberoamericana S. A. 1ª Edición. EUA. Año 1994.
4	Joyanes, L. **Programación en Tubo Pascal Versiones 5.5, 6.0, y 7.0**. Editorial *McGraw-Hill* / Interamericana de España, S. A. 2ª Edición. México. Año 1995.
5	Manuales de Referencia de Java, <**http://www.*sun*.com/java**>.

3.11. Hand Book

El *Hand Book* de IPC1 contiene los ejercicios y prácticas que son citadas en el IG. El *Hand Book* se desplegará en forma de imágenes. Éste contiene dos secciones: ejercicios y prácticas preparatorias.

Figura 2. Páginas 1 – 4 *Hand Book* “Programación principiantes”



C770 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1 CLASE HANDBOOK

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE CIENCIAS Y SISTEMAS

HAND BOOK
CONTENIDOS

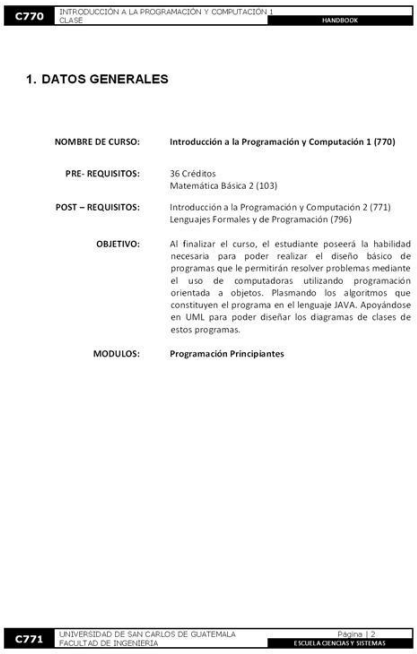
NOMBRE DE CURSO: Introducción a la Programación y Computación 1 CLASE

CODIGO DEL CURSO: 0770

NOMBRE DEL MODULO: Programación Principiantes

NO.	TEMAS	PAGINA
1	DATOS GENERALES	2
2	EJERCICIOS	3
3	PRÁCTICAS PREPARATORIAS	7

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.1 ESCUELA CIENCIAS Y SISTEMAS



C770 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1 CLASE HANDBOOK

1. DATOS GENERALES

NOMBRE DE CURSO: Introducción a la Programación y Computación 1 (770)

PRE- REQUISITOS: 36 Créditos
Matemática Básica 2 (103)

POST – REQUISITOS: Introducción a la Programación y Computación 2 (771)
Lenguajes Formales y de Programación (796)

OBJETIVO: Al finalizar el curso, el estudiante poseerá la habilidad necesaria para poder realizar el diseño básico de programas que le permitirán resolver problemas mediante el uso de computadoras utilizando programación orientada a objetos. Plasmando los algoritmos que constituyen el programa en el lenguaje JAVA. Apoyándose en UML para poder diseñar los diagramas de clases de estos programas.

MODULOS: Programación Principiantes

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.2 ESCUELA CIENCIAS Y SISTEMAS

C770 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1 CLASE HANDBOOK

2. EJERCICIOS

NOMBRE DE CURSO: Introducción a la Programación y Computación 1 CLASE

NOMBRE DEL MODULO: Programación Principiantes (1 sesión de ejercicios)

EJERCICIO SESIÓN NO. 27:	SUB-MÓDULO: INTRODUCCIÓN
OBJETIVO:	Ejercitar sesiones 1 - 26.
CONTENIDO:	Instrucciones: 1. Diseñe una jerarquia de clases y determine cuales de las clases podrian ser de tipo abstractas (defina las clases, sus atributos), utilice estructuras de datos para almacenar información: a. Una colección de CD. los cuales pueden ser de audio, con archivos mp3, de aplicaciones y de datos. Estos CD poseen nombre, dependiendo de su tipo particularidades especiales. 2. A partir del inciso 1.a. realizar los siguientes métodos: a. Ingresar, eliminar y modificar un CD. b. Buscar y mostrar los datos concernientes a un CD especifico (por medio de un nombre). c. Mostrar todos los CD agrupándolos por tipo. d. Mostrar todos los CD de aplicación. e. Mostrar los CD de un tipo especifico. f. Cuando finalice el programa este debe guardar la información de las estructuras de datos en un archivo de texto y al iniciarla se deben cargar los datos guardados en el archivo a memoria.

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.3 ESCUELA CIENCIAS Y SISTEMAS

C770 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1 CLASE HANDBOOK

3. PRÁCTICAS PREPARATORIAS

NOMBRE DE CURSO: Introducción a la Programación y Computación 1 CLASE

NOMBRE DEL MODULO: Programación Principiantes (5 prácticas)

PRÁCTICA PREPARATORIA NO. 1:	SUB-MÓDULO: PROGRAMACIÓN ORIENTADA A OBJETOS
OBJETIVO:	Ejercitar sesión 5-10.
CONTENIDO:	Instrucciones: A partir de los enunciados realizar lo que se solicita. 1. Definir una clase, la cual represente un automóvil. Su definición debe incluir: a. El modelo b. El color c. La matricula d. Año de fabricación e. Número de puertas f. Tipo de automóvil g. Si posee bolsa de aire para todos los pasajeros Definir los métodos que considere apropiados para esta clase. 2. A partir del concepto de figuras geométricas tridimensionales definir una clase abstracta que contenga al Cubo, Cono y Esfera. Definir atributos y métodos apropiados para cada clase. Explicar cómo utilizaría el concepto de polimorfismo para los métodos dibujar área y volumen de las figuras. 3. Realizar lo mismo que el inciso anterior sustituyendo la clase abstracta por una interfaz.

PRÁCTICA PREPARATORIA NO. 2:	SUB-MÓDULO: ELEMENTOS BÁSICOS DE PROGRAMACIÓN Y PROGRAMACIÓN ESTRUCTURADA
OBJETIVO:	Ejercitar sesiones 12-17.

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.4 ESCUELA CIENCIAS Y SISTEMAS

Figura 3. Páginas 5 – 6 *Hand Book* “Programación principiantes”

C770 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1 — CLASE — HANDBOOK

CONTENIDO:	**Instrucciones:** **A partir de los enunciados realizar lo que se solicita.** 1. **Realizar una clase que represente un semáforo, el cual puede estar rojo, amarillo o verde. En un principio el semáforo se encuentra en rojo y un atributo indica si se encuentra parpadeando o no. Y posee un método que le permite cambiar de color y otro que le permite establecer si parpadea. Asimismo un método que verifica si el semáforo se encuentra en rojo mostrar una advertencia de alto. Si el semáforo se encuentra en verde que avance. Si el semáforo se encuentra en verde y parpadeando despliegar una advertencia de cuidado, esta misma advertencia se despliega si el semáforo se encuentra en amarillo.** 2. **Realizar una clase estableciendo sus atributos y métodos que permita realizar las tablas de multiplicar del número 6 al 8. (Realizar esto utilizando estructuras de control for, while y do while).** 3. **Realizar un método recursivo (si es necesario añada más métodos), que permita obtener el número de combinaciones de m objetos en n, utilizando el triángulo de Pascal:** 1 1 1 1 2 1 1 3 3 1 1 4 6 4 1 . . . . . . **Donde $_nC_m = {}_{n-1}C_{m-1} + {}_{n-1}C_m$ para toda $0 < m < n$.**

PRÁCTICA PREPARATORIA NO. 3:	SUB-MÓDULO: ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCION
OBJETIVO:	Ejercitar sesiones 19-20.
CONTENIDO:	**Instrucciones:** **A partir del enunciado escribir lo que se solicita:** 1. **Escribir una clase cuyo método main() implemente un programa que realice:** a. **Declarar un array de enteros, que almacene los números del 0 al 9 en orden creciente.** b. **Declarar un array de tipo char, que contenga las 5 vocales en mayúscula.** c. **Concatenar los caracteres del array de 5 vocales en una variable tipo String.** d. **Mostrar en pantalla el contenido del array de vocales y de la variable tipo String.** e. **Mostrar en pantalla la suma del array de enteros.** 2. **Escribir un método, que reciba por parámetro un array de enteros y que muestre sus valores en pantalla (uno por línea).** 3. **Escribir un método, que reciba por parámetro un array de caracteres y devuelva una cadena con su contenido.** 4. **Implemente los algoritmos de ordenamiento de datos en arreglos. (Intercambio, selección, inserción, burbuja y quicksort).**

PRÁCTICA PREPARATORIA NO. 4:	SUB-MÓDULO: FLUJOS (STREAMS) Y MANIPULACIÓN DE ARCHIVOS

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA — FACULTAD DE INGENIERÍA — Página 1.5 — ESCUELA CIENCIAS Y SISTEMAS

C770 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1 — CLASE — HANDBOOK

OBJETIVO:	Ejercitar sesiones 21-22.
CONTENIDO:	**Instrucciones:** **A partir del enunciado escribir lo que se solicita:** 1. **Realice una clase donde se pueda realizar la escritura y lectura de archivos, utilizando las clases FileInputStream, DataInputStream, FileOutputStream, DataOutputStream, RandomAccessFile. Deben existir métodos para poder escribir una cadena de caracteres (String) en cada uno de los 3 tipos de archivos (FileStream, DataStream y RandomAccessStream) así mismo se debe tener métodos que lean desde cada uno de los 3 tipos de archivos y despliegue su contenido en pantalla.** 2. **A partir del documento “Cusumano Cusumano - Technology Strategy and Management.pdf”, que se encuentra dentro de la carpeta “crpDocs” dentro de la carpeta que contiene este Handbook, realizar un resumen del mismo e indicar al finalizar sus opiniones al respecto sobre lo planteado en el mismo.**

PRÁCTICA PREPARATORIA NO. 5:	SUB-MÓDULO: TIPO DE DATOS ABSTRACTOS
OBJETIVO:	Ejercitar sesiones 24-26.
CONTENIDO:	**Instrucciones:** **Realice lo que a continuación se solicita:** 1. **Escribir un programa que utilice una pila de caracteres para procesar cada carácter de una expresión que viene dada en una línea y permita la verificación del equilibrio de paréntesis, corchetes y llaves {},[],().** **Ejemplo de expresión en equilibrio ((x+y)/4)*8. Ejemplo de expresión en desequilibrio: 3*[(8-4)*3+a-b/7, a esta expresión le falta un corchete.** 2. **Se desea identificar si una frase es palindroma o no, por lo cual se necesita hacer un programa que solicite la frase y posteriormente utilice una pila y una cola para poder determinar si la frase es palindroma. Se sugiere añadir cada carácter de la frase a una pila y a la vez a una cola, extraer carácter a carácter simultáneamente de la pila y de la cola y su comparación determinará si es palindroma o no.** 3. **A partir del documento “Cusumano - How Microsoft Build Software.pdf”, que se encuentra dentro de la carpeta “crpDocs” dentro de la carpeta que contiene este Handbook, realizar un resumen del mismo e indicar al finalizar sus opiniones al respecto sobre lo planteado en el mismo.**

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA — FACULTAD DE INGENIERÍA — Página 1.6 — ESCUELA CIENCIAS Y SISTEMAS

4. GUÍA DEL INSTRUCTOR DE LA CLASE DE INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2

La guía del instructor del curso de Introducción a la Programación y Computación 2, se realizó conforme al formato sugerido en el curso de "Estructuración de cursos y laboratorios", impartido por el señor Mruntunjaya Panda.

4.1. Datos generales

NOMBRE DE CURSO:	**Introducción a la Programación y Computación 2 (771)**
PRE- REQUISITOS:	Introducción a la Programación y Computación 1 (770)
	Matemática Intermedia 1 (107)
	Lógica de Sistemas (795)
	Matemática de Cómputo 1 (960)
POST – REQUISITOS:	Introducción a la Programación y Computación 2 (771)

Lenguajes Formales y de Programación (796)

OBJETIVO: Al finalizar el curso, el estudiante poseerá la habilidad necesaria para poder realizar el diseño básico de programas, que le permitirán resolver problemas mediante el uso de computadoras, utilizando programación orientada a objetos. Plasmando los algoritmos que constituyen el programa en el lenguaje JAVA. Apoyándose en UML para poder diseñar los diagramas de clases de estos programas.

NOMBRE DEL MÓDULO: Programación intermedios

VIGENCIA: Primer Semestre 2009

4.2. Distribución del curso

El curso de Introducción a la Programación y Computación 2 se divide en veintiocho sesiones, integradas en un módulo denominado Programación intermedios. La tabla XIII muestra la tabla correspondiente a la distribución del curso de IPC2.

Tabla XIII. Distribución del curso de IPC2

MÓDULO	TEORÍA	PRÁCTICA	TOTAL	EXAMEN	PRÁCTICA PREPARATORIA	PROYECTOS
PROGRAMACIÓN INTERMEDIOS	38 horas	18 horas	56 horas	3	3	3
TOTAL	38 horas	18 horas	56 horas	3	3	3

4.3. Distribución del módulo

El módulo de programación intermedios se divide en cinco sub – módulos. La tabla XIV muestra la tabla correspondiente a la distribución del módulo del curso de IPC2.

Tabla XIV. Distribución del módulo de IPC2

SUB-MÓDULO	TEORÍA	PRÁCTICA	TOTAL	AL FINALIZAR LOS SUB-MÓDULOS EL ESTUDIANTE SERA CAPAZ DE
INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES	4 horas	4 horas	8 horas	Diseñar modelos simples de bases de datos relacionales.
METODOLOGÍA PARA DESARROLLO DE SOFTWARE	8 horas	0 horas	8 horas	Conocer el proceso de desarrollo de *software* y los conceptos fundamentales de UML.

ETAPA DE ANÁLISIS DEL CICLO DE CONSTRUCCIÓN	14 horas	6 horas	20 horas	Identificar y realizar la estructura estática de un sistema de *software*.
ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCION	6 horas	4 horas	10 horas	Utilizar diagramas de clases para modelar la estructura estática de un sistema de *software*.
INTEGRACION DE ARTEFACTOS DE SOFTWARE	6 horas	4 horas	10 horas	Realizar el modelado físico de un sistema.
TOTAL	38 horas	18 horas	56 oras	

4.4. Evaluación

Ponderación de evaluaciones, tareas, prácticas y laboratorio. La tabla XV muestra la tabla correspondiente al detalle de evaluación del curso de IPC2.

Tabla XV. Distribución de la evaluación de IPC2

ACTIVIDAD	NÚMERO	PORCENTAJE	TOTAL
PRÁCTICA PREPARATORIA	3	1.00 %	3.00 %
AUTO-EVALUACIÓN	2	1.00 %	2.00 %

LABORATORIO	1	30.0 %	30.00 %
EXAMEN PARCIAL	2	20.0 %	40.00 %
EXAMEN FINAL	1	25.0 %	25.00 %
		TOTAL	**100.0**

4.5. Detalle de sesiones

La tabla XVI muestra la tabla correspondiente al detalle de las sesiones del curso de IPC2.

Tabla XVI. Detalle de sesiones de IPC2

NO	TEORÍA	NO	PRÁCTICA	PRÁCTICA PREPARATORIA / PROYECTO
1	**PROGRAMACIÓN INTERMEDIOS** – 1	2	**PROGRAMACIÓN INTERMEDIOS** – 2	
3	**PROGRAMACIÓN INTERMEDIOS** – 3	4	**PROGRAMACIÓN INTERMEDIOS** – 4	Practica No.1
5	**PROGRAMACIÓN INTERMEDIOS** – 5			
6	**PROGRAMACIÓN INTERMEDIOS** – 6			
7	**PROGRAMACIÓN**			

	INTERMEDIOS – 7			
8	**PROGRAMACIÓN INTERMEDIOS** – 8			
9	**PROGRAMACIÓN INTERMEDIOS** – 9			
10	**PROGRAMACIÓN INTERMEDIOS** – 10	11	**PROGRAMACIÓN INTERMEDIOS** – 11	
12	**PROGRAMACIÓN INTERMEDIOS** – 12			
13	**PROGRAMACIÓN INTERMEDIOS** – 13			
14	**PROGRAMACIÓN INTERMEDIOS** – 14	15	**PROGRAMACIÓN INTERMEDIOS** – 15	Práctica No. 2
17	**PROGRAMACIÓN INTERMEDIOS** – 17	16	**PROGRAMACIÓN INTERMEDIOS** – 16	
18	**PROGRAMACIÓN INTERMEDIOS** – 18			
19	**PROGRAMACIÓN INTERMEDIOS** – 19			
20	**PROGRAMACIÓN INTERMEDIOS** – 20			
21	**PROGRAMACIÓN INTERMEDIOS** – 21	23	**PROGRAMACIÓN INTERMEDIOS** – 23	Práctica No. 3
22	**PROGRAMACIÓN INTERMEDIOS** – 22	24	**PROGRAMACIÓN INTERMEDIOS** – 24	
25	**PROGRAMACIÓN INTERMEDIOS** – 25	27	**PROGRAMACIÓN INTERMEDIOS** – 27	
26	**PROGRAMACIÓN INTERMEDIOS** – 26	28	**PROGRAMACIÓN INTERMEDIOS** – 28	

4.6. Distribución de sesiones

La tabla XVII muestra la tabla correspondiente a la distribución de sesiones del curso de IPC2.

Tabla XVII. Distribución de sesiones de IPC2

SESIÓN NO. 1:	**SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES**
OBJETIVO:	**Introducir a los estudiantes los conceptos básicos de las bases de datos relacionales.**
CONTENIDO:	**1. Conceptos Básicos de Bases de Datos** **1.1. Modelo** **1.2. Base de Datos** **1.3. Sistema Administrador de Base de Datos** **1.3.1. Usuario** **1.3.2. Arquitectura de 3 Capas** **1.4. Base de Datos Relacional** **1.4.1. Modelo Relacional** **1.4.2. Entidad** **1.4.3. Atributos** **1.4.4. Tupla** **1.4.5. Llaves Primarias** **1.4.6. Llaves Foráneas**

SESIÓN NO. 2:	**SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES**
OBJETIVO:	**Identificar llaves primarias entre diversas entidades.**
CONTENIDO:	**1. Ver ejercicio sesión no. 2 en el *Handbook*.**
SESIÓN NO. 3:	**SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES**
OBJETIVO:	**Introducir a los estudiantes los conceptos de relaciones básicas utilizados en las bases de datos relacionales.**
CONTENIDO:	**1. Conceptos Básicos de Bases de Datos** **1.1. Relaciones básicas entre entidades** **1.1.1. De uno a uno** **1.1.2. De uno a muchos** **1.1.3. De muchos a muchos** **1.1.4. Opcionalidad en las relaciones** **1.1.5. Relaciones como UID** **1.1.6. Eliminar relación muchos a muchos** **1.2. Diagrama Entidad Relación** **1.2.1. Reglas de Entidades** **1.2.2. Reglas de Atributos** **1.3. Diseño de Base de Datos** **1.3.1. Mapeo Conceptual** **1.3.2. Esquema Conceptual**
SESIÓN NO. 4:	**SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES**

OBJETIVO:	**Realizar un diagrama entidad relación para aplicar los conceptos de relación y llaves foráneas.**
CONTENIDO:	**1. Ver ejercicio sesión no. 4 en el *Handbook*.**
SESIÓN NO. 5:	**SUB-MÓDULO: METODOLOGIA PARA DESARROLLO DE *SOFTWARE***
OBJETIVO:	**Adquirir los conocimientos generales respecto a metodologías de software.**
CONTENIDO:	**1. Métodos de Programación** **2. Introducción al Proceso de Desarrollo de *Software*** **2.1. Aspectos** **2.2. Fases Generales** **2.3. Plan**
SESIÓN NO. 6:	**SUB-MÓDULO: METODOLOGIA PARA DESARROLLO DE *SOFTWARE***
OBJETIVO:	**Introducir a los estudiantes al lenguaje unificado de modelado (UML), presentando las ventajas para el diseño de *software*.**
CONTENIDO:	**1. Introducción a UML – Lenguaje Unificado de Modelado** **1.1. Antecedentes** **1.2. Modelado Visual** **1.3. Definición** **1.4. Objetivos** **1.5. Bloques de Construcción** **1.6. Ventajas** **1.7. Vistas**

	1.8. Diagramas
SESIÓN NO. 7:	**SUB-MÓDULO: METODOLOGIA PARA DESARROLLO DE *SOFTWARE***
OBJETIVO:	**Conocer las etapas de la metodología iterativa incremental, obteniendo conceptos generales.**
CONTENIDO:	**1. Análisis y Diseño Orientado a Objetos** **1.1. Conceptos Básicos** **1.1.1. Objeto** **1.1.2. Tipos de Objeto** **1.1.3. Entidad – Objeto** **1.1.4. Clase** **1.1.5. Método** **1.1.6. Encapsulado** **1.1.7. Mensajes** **1.1.8. Herencia** **1.1.9. Polimorfismo** **1.2. Conceptos Generales** **1.2.1. Análisis** **1.2.2. Diseño** **1.2.3. Análisis Orientado a Objetos** **1.2.4. Diseño Orientado a Objetos** **1.2.5. Equipo de Trabajo** **2. Proceso de Desarrollo de *Software*** **2.1. Método Iterativo Incremental** **2.1.1. Pasos a Macro nivel**
SESIÓN NO. 8:	**SUB-MÓDULO: METODOLOGIA PARA DESARROLLO DE *SOFTWARE***
	Aprender sobre las fases fundamentales de la metodología iterativa

OBJETIVO:	**incremental y sobre sus aplicaciones.**
CONTENIDO:	**1. Proceso de Desarrollo de *Software*** **1.1. Método Iterativo Incremental** **1.1.1. Fase de Planeación y Elaboración** **1.1.2. Ciclos de Desarrollo** **1.1.2.1. Perfeccionamiento del Plan** **1.1.2.2. Sincronización de Artefactos** **1.1.2.3. Análisis** **1.1.2.4. Diseño** **1.1.2.5. Construcción (programación)** **1.1.2.6. Pruebas** **1.1.3. Aplicación** ***1.2.* Mejores Prácticas para Desarrollar *Software*** **1.3. Capas Arquitectónicas** **1.3.1. Capa de Presentación** **1.3.2. Capa de Aplicación** **1.3.3. Capa de Datos** **1.3.4. Requerimientos** **a. Artefactos de Soporte para Requerimientos**
SESIÓN NO. 9:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Aprender sobre la utilización y la importancia de los casos de uso en el desarrollo de *software*.**
CONTENIDO:	**1. Casos de Uso** **1.1. Introducción** **1.1.1. Definición** **1.1.2. Notación** **1.1.3. Modalidades Generales de Casos de Uso** **1.1.3.1. CDU de Alto Nivel** **1.1.3.2. Diagramas CDU** **1.1.3.3. CDU Expandidos**

	1.1.4. Tipos de Casos de Uso **1.2. Actores** **1.2.1. Definición** **1.2.2. Notación**
SESIÓN NO. 10:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Aprender sobre la forma de realizar diagramas de casos de uso y de las relaciones importantes entre los casos de uso.**
CONTENIDO:	**1. Casos de Uso** **1.1. Identificación de Casos de Uso** **1.1.1. Introducción** **1.1.2. Métodos de identificación** **1.2. Diagramas de Casos de Uso** **1.2.1. Consideraciones** **1.2.2. Diagrama** **1.2.3. Ejemplos** **1.3. Sistemas y sus Fronteras** **1.3.1. Fronteras Ordinarias** **1.3.2. Utilización de CDU** **1.3.3. Ventajas** **1.4. Relaciones en un diagrama de casos de uso** ***1.4.1. Communicates*** ***1.4.2. Uses*** ***1.4.3. Extend*** **1.4.4. Generalización** **1.5. Ejemplo**
SESIÓN NO. 11:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Ejercitar sesiones 9-10**

CONTENIDO:	**1. Ver ejercicio sesión no. 11 en el *Handbook.***
SESIÓN NO. 12:	**SUB-MÓDULO: ETAPA DE ANÁLISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Aprender los conceptos fundamentales relacionados con diagramas de clase y su importancia en el desarrollo de *software*.**
CONTENIDO:	**1. Diagramas de Estructura Estática** **1.1. Diagrama de Clases** **1.1.1. Clase** **1.1.2. Atributo** **1.1.3. Operación** **1.1.4. Método** **1.1.5. Objeto (instancia)** **1.1.6. Notación**
SESIÓN NO. 13:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Presentar a los estudiantes las distintas relaciones que pueden surgir al momento de realizar diagramas de clases.**
CONTENIDO:	**1. Diagramas de Estructura estática** **1.1. Diagrama de Clases** **1.1.1. Relaciones entre clases** **1.1.1.1. Asociación** **1.1.1.1.1. Binaria** **1.1.1.1.2. N-aria** **1.1.1.2. Composición** **1.1.1.3. Generalización**

	1.1.1.4. Dependencia **1.1.2. Notas**
SESIÓN NO. 14:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Presentar a los estudiantes estrategias sobre la identificación de los conceptos al momento de realizar diagramas de clases.**
CONTENIDO:	**1. Diagramas de Estructura estática** **1.1. Estrategia para Identificar conceptos** **1.1.1. Directrices para construir el modelo conceptual** **1.1.2. Asignación de nombres y modelado de los conceptos** **1.1.3. Agregación de relaciones** **1.1.4. Categorías de alta prioridad** **1.1.5. Directrices para identificar asociaciones**
SESIÓN NO. 15:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Ejercitar sesiones 12-14**
CONTENIDO:	**1. Ver ejercicio sesión no. 15 en el *Handbook*.**
SESIÓN NO. 16:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Comprobar los conocimientos adquiridos por los alumnos a partir de la sesión 1 hasta la sesión 14.**
CONTENIDO:	**1. Examen Parcial número 1**

SESIÓN NO. 17:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Presentar la importancia de definir un glosario y de realizar diagramas de secuencia después de realizar diagramas de caso de uso.**
CONTENIDO:	**1. Definición de Glosario** **1.1. Definición** **1.2. Estructura** **2. Comportamiento inicial del sistema** **2.1. Introducción** **2.2. Introducción a diagramas de secuencia** **2.2.1. Características** **2.2.2. Elaboración diagrama de secuencia para el curso normal de casos de uso** **2.2.3. Asignación de nombres a eventos de un sistema** **2.2.4. Notación** **2.2.5. Línea de vida** **2.2.6. Activación** **2.2.7. Mensaje** **2.2.8. Ejemplo**
SESIÓN NO. 18:	**SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN**
OBJETIVO:	**Introducir a los estudiantes sobre la importancia de los diagramas de estado.**
CONTENIDO:	**1. Comportamiento inicial del sistema** **1.1. Introducción a diagramas de estado** **1.1.1. Características** **1.1.2. Estado** **1.1.3. Eventos** **1.1.4. Representación** **1.1.5. Envío de mensajes**

	1.1.6. Transición simple **1.1.7. Transición interna** **1.1.8. Tipos independientes y dependientes del estado** **1.1.9. Tipos y clases comúnmente dependientes del estado** **1.1.10. Tipos de Eventos**
SESIÓN NO. 19:	**SUB-MÓDULO: ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCION**
OBJETIVO:	**Introducir a los estudiantes en los conceptos básicos del ciclo de construcción.**
CONTENIDO:	**1. Interfaz de usuario** **1.1. Reportes** **1.2. Secuencia de Pantallas** **2. Diagrama de Clases** **2.1. Definición** **2.2. Dependencias** **2.2.1. Modelo Conceptual** **2.2.2. Diagramas de Iteración** **2.3. Relaciones de Dependencia** **2.4. Clase Controladora** **2.5. Clases Abstractas**
SESIÓN NO. 20:	**SUB-MÓDULO: ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCION**
OBJETIVO:	**Definir y presentar lo que es el diagrama de clases aplicado a los métodos y paquetes y repasar los conceptos fundamentales de la metodología de desarrollo.**
CONTENIDO:	**1. Diagramas de Clase** **1.1. Cómo elaborar un Diagrama de Clase** **1.2. Métodos** **1.3. Paquetes**

	2. Repaso Metodología de Desarrollo **2.1. Planeación y Elaboración** **2.2. Construcción Ciclos de Desarrollo** **2.3. Aplicación** **2.4. Documentación Técnica**
SESIÓN NO. 21:	**SUB-MÓDULO: ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCION**
OBJETIVO:	**Aprender la importancia de realizar un diagrama de colaboración en la construcción de un producto de software.**
CONTENIDO:	**1. Introducción a Diagrama de Colaboración** **1.1. Dependencias** **1.2. Diagramas de Interacción** **1.3. Características Principales** **1.4. Objeto** **1.5. Enlaces** **1.6. Preparación de Diagramas de Colaboración** **1.7. Flujo de Mensajes** **1.8. Sintaxis para Mensajes** **1.9. Iteración o Ciclo** **1.10. Marcadores de creación y destrucción de Objetos** **1.11. Ejemplo**
SESIÓN NO. 22:	**SUB-MÓDULO: ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCION**
OBJETIVO:	**Ejercitar sesiones 17-21.**
CONTENIDO:	**1. Ver ejercicio sesión no. 22 en el *Handbook*.**
SESIÓN NO. 23:	**SUB-MÓDULO: ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCION**

OBJETIVO:	**Comprobar los conocimientos adquiridos por los alumnos a partir de la sesión 16 hasta la sesión 23.**
CONTENIDO:	**1. Examen Parcial número 2**
SESIÓN NO. 24:	**SUB-MÓDULO: INTEGRACION DE ARTEFACTOS DE *SOFTWARE***
OBJETIVO:	**Presentar en qué consiste un diagrama de actividades y cuál es su función en la construcción de un producto de software.**
CONTENIDO:	**1. Introducción a diagramas de actividades** **1.1. Características** **1.2. Estado de Acción** **1.3. Estado de Actividad** **1.4. Transiciones** **1.5. Bifurcaciones** **1.6. Ejemplos**
SESIÓN NO. 25:	**SUB-MÓDULO: INTEGRACION DE ARTEFACTOS DE *SOFTWARE***
OBJETIVO:	**Aprender en qué consiste el modelado físico de un sistema aprendiendo el diagrama de componentes y de despliegue.**
CONTENIDO:	**1. Modelado Físico de un Sistema** **1.1. Componente** **1.1.1. Características** **1.1.2. Notación** **1.2. Interfaces** **1.2.1. Definición** **1.2.2. Notación**

	1.3. Tipos de Componentes **1.3.1.1. De Despliegue** **1.3.1.2. Producto del Trabajo** **1.3.1.3. De Ejecución** **1.4. Organización de Componentes** **1.5. Estereotipos de Componentes** **1.6. Nodo** **1.6.1. Definición** **1.6.2. Características** **1.6.3. Notación**
SESIÓN NO. 26:	**SUB-MÓDULO: INTEGRACION DE ARTEFACTOS DE SOFTWARE**
OBJETIVO:	**Introducir a los estudiantes a los diagramas de implementación.**
CONTENIDO:	**1. Introducción a los diagramas de implementación** **1.1. Diagrama de Componentes** **1.1.1. Definición** **1.1.2. Representación** **1.2. Diagrama de Despliegue** **1.2.1. Definición** **1.2.2. Representación**
SESIÓN NO. 27:	**SUB-MÓDULO: INTEGRACION DE ARTEFACTOS DE *SOFTWARE***
OBJETIVO:	**Ejercitar sesiones 24 - 26.**
CONTENIDO:	**1. Ver ejercicio sesión no. 27 en el *Handbook*.**

SESIÓN NO. 28:	**SUB-MÓDULO: INTEGRACION DE ARTEFACTOS DE *SOFTWARE***
OBJETIVO:	**Comprobar los conocimientos adquiridos por los alumnos a partir de la sesión 1 hasta la sesión 27.**
CONTENIDO:	**1. Examen Final**

4.7. Detalle de conducción de tutoriales

La tabla XVIII muestra la tabla correspondiente a la conducción de tutoriales del curso de IPC2.

Tabla XVIII. Detalle de conducción de tutoriales de IPC2

ACTIVIDAD	DETALLE
Iniciado en sesión de clase número:	N.A.
Finalizado en sesión de clase número	N.A.
Viva / *Demo* para ser conducido:	N.A.
Criterio de evaluación:	N.A.
Viva:	N.A.

Documentación:	N.A.

4.8. Detalle de conducción de exámenes

La tabla XIX muestra la tabla correspondiente al detalle de conducción de exámenes del curso de IPC2.

Tabla XIX. Detalle de conducción de exámenes de IPC1

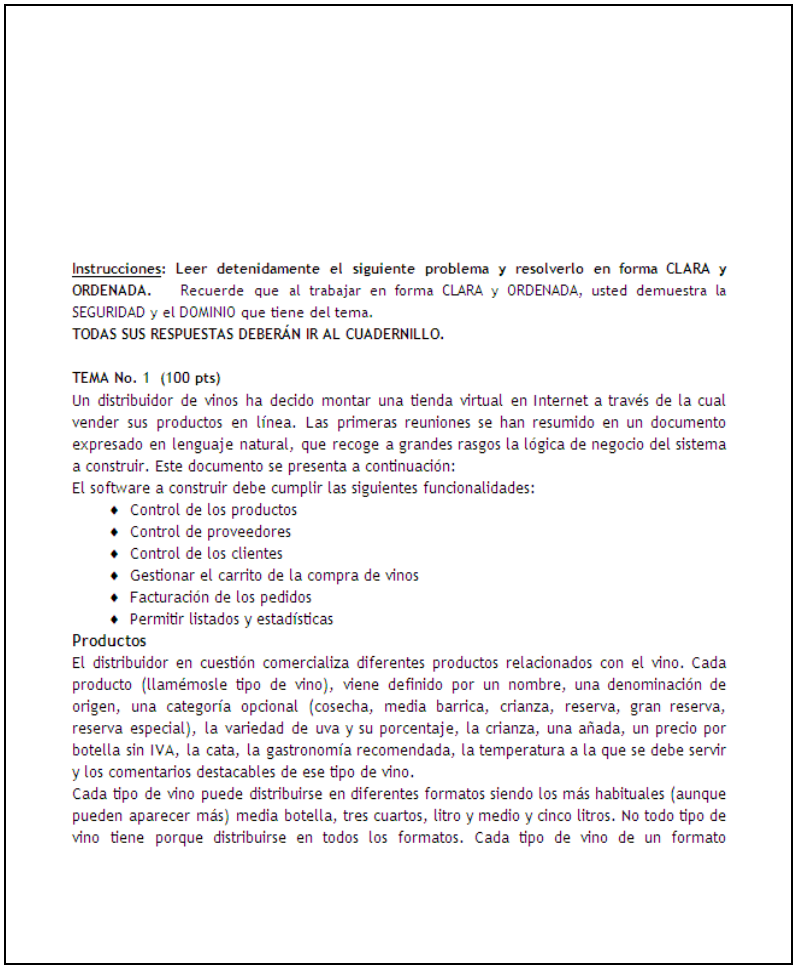
EXAMEN NÚMERO:	**1**
Examen a ser realizado después de la sesión número:	15
Temas para el examen:	Sesiones: 1 – 15
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%
EXAMEN NÚMERO:	**2**
Examen a ser realizado después de la sesión número:	23
Temas para el examen:	Sesiones: 16 – 23
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%

EXAMEN NÚMERO:	3
Examen a ser realizado después de la sesión número:	27
Temas para el examen:	Sesiones: 1 – 27, haciendo especial énfasis en las sesiones 25 – 27
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%

4.9. Ejemplo de examen

La Figura 4 muestra el examen final de ejemplo del curso de IPC2.

Figura 4. Examen final de ejemplo de IPC2

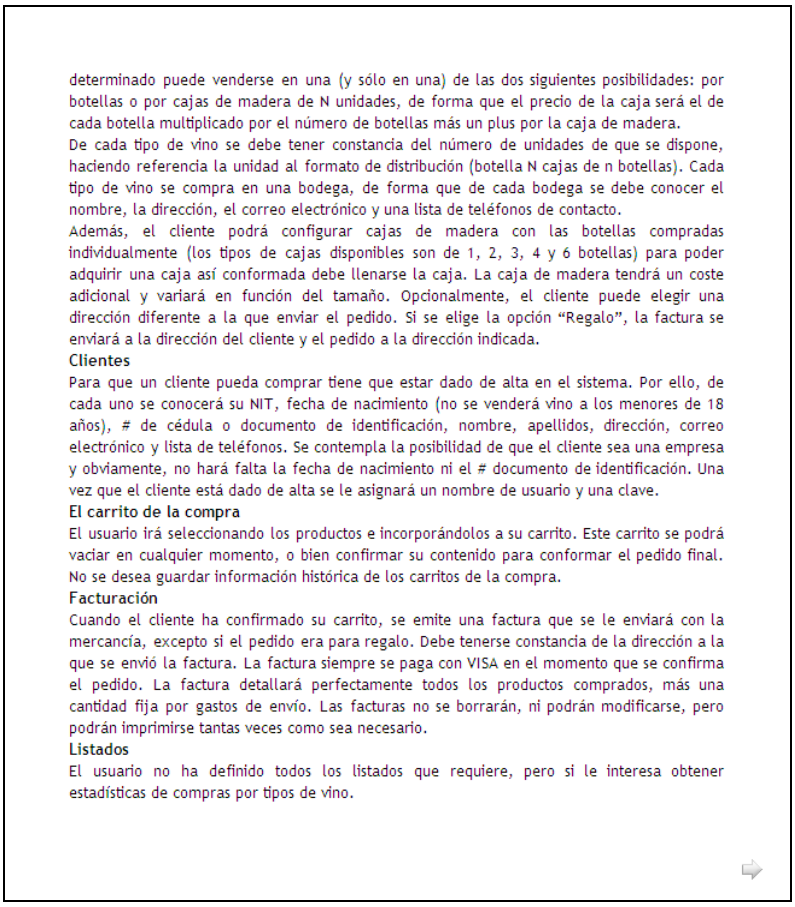


Instrucciones: **Leer detenidamente el siguiente problema y resolverlo en forma CLARA y ORDENADA.** Recuerde que al trabajar en forma CLARA y ORDENADA, usted demuestra la SEGURIDAD y el DOMINIO que tiene del tema.
TODAS SUS RESPUESTAS DEBERÁN IR AL CUADERNILLO.

TEMA No. 1 (100 pts)
Un distribuidor de vinos ha decido montar una tienda virtual en Internet a través de la cual vender sus productos en línea. Las primeras reuniones se han resumido en un documento expresado en lenguaje natural, que recoge a grandes rasgos la lógica de negocio del sistema a construir. Este documento se presenta a continuación:
El software a construir debe cumplir las siguientes funcionalidades:

- Control de los productos
- Control de proveedores
- Control de los clientes
- Gestionar el carrito de la compra de vinos
- Facturación de los pedidos
- Permitir listados y estadísticas

Productos
El distribuidor en cuestión comercializa diferentes productos relacionados con el vino. Cada producto (llamémosle tipo de vino), viene definido por un nombre, una denominación de origen, una categoría opcional (cosecha, media barrica, crianza, reserva, gran reserva, reserva especial), la variedad de uva y su porcentaje, la crianza, una añada, un precio por botella sin IVA, la cata, la gastronomía recomendada, la temperatura a la que se debe servir y los comentarios destacables de ese tipo de vino.
Cada tipo de vino puede distribuirse en diferentes formatos siendo los más habituales (aunque pueden aparecer más) media botella, tres cuartos, litro y medio y cinco litros. No todo tipo de vino tiene porque distribuirse en todos los formatos. Cada tipo de vino de un formato



determinado puede venderse en una (y sólo en una) de las dos siguientes posibilidades: por botellas o por cajas de madera de N unidades, de forma que el precio de la caja será el de cada botella multiplicado por el número de botellas más un plus por la caja de madera.
De cada tipo de vino se debe tener constancia del número de unidades de que se dispone, haciendo referencia la unidad al formato de distribución (botella N cajas de n botellas). Cada tipo de vino se compra en una bodega, de forma que de cada bodega se debe conocer el nombre, la dirección, el correo electrónico y una lista de teléfonos de contacto.
Además, el cliente podrá configurar cajas de madera con las botellas compradas individualmente (los tipos de cajas disponibles son de 1, 2, 3, 4 y 6 botellas) para poder adquirir una caja así conformada debe llenarse la caja. La caja de madera tendrá un coste adicional y variará en función del tamaño. Opcionalmente, el cliente puede elegir una dirección diferente a la que enviar el pedido. Si se elige la opción "Regalo", la factura se enviará a la dirección del cliente y el pedido a la dirección indicada.
Clientes
Para que un cliente pueda comprar tiene que estar dado de alta en el sistema. Por ello, de cada uno se conocerá su NIT, fecha de nacimiento (no se venderá vino a los menores de 18 años), # de cédula o documento de identificación, nombre, apellidos, dirección, correo electrónico y lista de teléfonos. Se contempla la posibilidad de que el cliente sea una empresa y obviamente, no hará falta la fecha de nacimiento ni el # documento de identificación. Una vez que el cliente está dado de alta se le asignará un nombre de usuario y una clave.
El carrito de la compra
El usuario irá seleccionando los productos e incorporándolos a su carrito. Este carrito se podrá vaciar en cualquier momento, o bien confirmar su contenido para conformar el pedido final. No se desea guardar información histórica de los carritos de la compra.
Facturación
Cuando el cliente ha confirmado su carrito, se emite una factura que se le enviará con la mercancía, excepto si el pedido era para regalo. Debe tenerse constancia de la dirección a la que se envió la factura. La factura siempre se paga con VISA en el momento que se confirma el pedido. La factura detallará perfectamente todos los productos comprados, más una cantidad fija por gastos de envío. Las facturas no se borrarán, ni podrán modificarse, pero podrán imprimirse tantas veces como sea necesario.
Listados
El usuario no ha definido todos los listados que requiere, pero si le interesa obtener estadísticas de compras por tipos de vino.

4.10. Bibliografía recomendada

La tabla XX muestra la tabla correspondiente a la bibliografía recomendada del curso de IPC2.

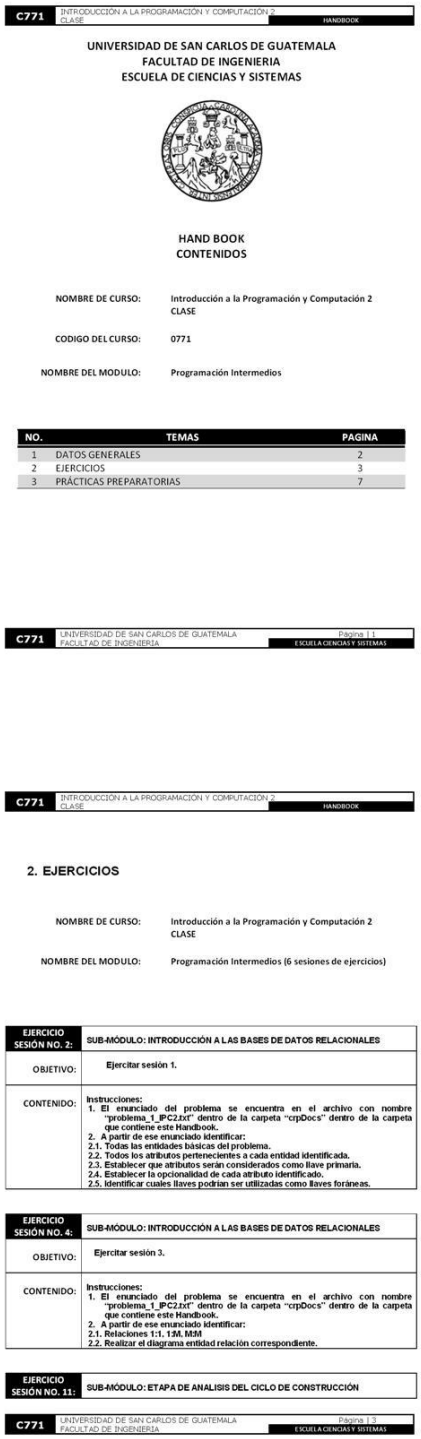
Tabla XX. Bibliografía recomendada de IPC2

No.	Libro
1	Craig Larman. **UML Y Patrones, Introducción al Análisis y Diseño Orientado a Objetos**. Editorial *Prentice Hall*. 2ª Edición. México. Año 2004.
2	Date, C. J. **Introducción a los sistemas de bases de datos.** Editorial *Prentice Hall*. 7ª. Edición. México. Año 2001.

4.11. Hand Book

El *Hand Book* de IPC2 contiene los ejercicios y prácticas que son citadas en el IG. El *Hand Book* se desplegará en forma de imágenes. Éste contiene dos secciones: Ejercicios y Prácticas Preparatorias.

Figura 5. Páginas 1 – 4 *Hand Book* “Programación intermedios”



C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2 CLASE HANDBOOK

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE CIENCIAS Y SISTEMAS

HAND BOOK
CONTENIDOS

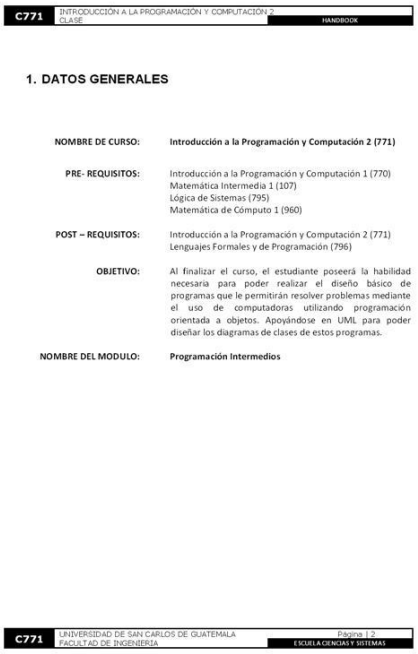
NOMBRE DE CURSO: Introducción a la Programación y Computación 2 CLASE

CODIGO DEL CURSO: 0771

NOMBRE DEL MODULO: Programación Intermedios

NO.	TEMAS	PAGINA
1	DATOS GENERALES	2
2	EJERCICIOS	3
3	PRÁCTICAS PREPARATORIAS	7

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.1 ESCUELA CIENCIAS Y SISTEMAS



C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2 CLASE HANDBOOK

1. DATOS GENERALES

NOMBRE DE CURSO: Introducción a la Programación y Computación 2 (771)

PRE- REQUISITOS: Introducción a la Programación y Computación 1 (770)
Matemática Intermedia 1 (107)
Lógica de Sistemas (795)
Matemática de Cómputo 1 (960)

POST – REQUISITOS: Introducción a la Programación y Computación 2 (771)
Lenguajes Formales y de Programación (796)

OBJETIVO: Al finalizar el curso, el estudiante poseerá la habilidad necesaria para poder realizar el diseño básico de programas que le permitirán resolver problemas mediante el uso de computadoras utilizando programación orientada a objetos. Apoyándose en UML para poder diseñar los diagramas de clases de estos programas.

NOMBRE DEL MODULO: Programación Intermedios

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.2 ESCUELA CIENCIAS Y SISTEMAS

C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2 CLASE HANDBOOK

2. EJERCICIOS

NOMBRE DE CURSO: Introducción a la Programación y Computación 2 CLASE

NOMBRE DEL MODULO: Programación Intermedios (6 sesiones de ejercicios)

EJERCICIO SESIÓN NO. 2:	SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES
OBJETIVO:	Ejercitar sesión 1.
CONTENIDO:	Instrucciones: 1. El enunciado del problema se encuentra en el archivo con nombre “problema_1_IPC2.txt” dentro de la carpeta “crpDocs” dentro de la carpeta que contiene este Handbook. 2. A partir de ese enunciado identificar: 2.1. Todas las entidades básicas del problema. 2.2. Todos los atributos pertenecientes a cada entidad identificada. 2.3. Establecer que atributos serán considerados como llave primaria. 2.4. Establecer la opcionalidad de cada atributo identificado. 2.5. Identificar cuales llaves podrían ser utilizadas como llaves foráneas.

EJERCICIO SESIÓN NO. 4:	SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES
OBJETIVO:	Ejercitar sesión 3.
CONTENIDO:	Instrucciones: 1. El enunciado del problema se encuentra en el archivo con nombre “problema_1_IPC2.txt” dentro de la carpeta “crpDocs” dentro de la carpeta que contiene este Handbook. 2. A partir de ese enunciado identificar: 2.1. Relaciones 1:1, 1:M, M:M 2.2. Realizar el diagrama entidad relación correspondiente.

EJERCICIO SESIÓN NO. 11:	SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.3 ESCUELA CIENCIAS Y SISTEMAS

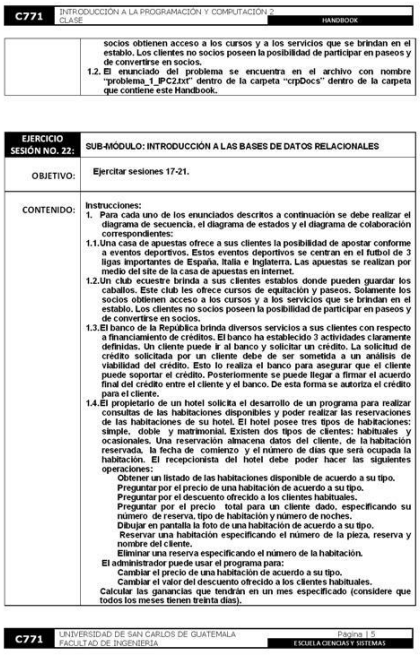
C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2 CLASE HANDBOOK

OBJETIVO:	Ejercitar sesiones 9-10.
CONTENIDO:	Instrucciones: 1. Para cada uno de los enunciados descritos a continuación se debe realizar el diagrama de caso de uso correspondiente: 1.1. Una casa de apuestas ofrece a sus clientes la posibilidad de apostar conforme a eventos deportivos. Estos eventos deportivos se centran en el futbol de 3 ligas importantes de España, Italia e Inglaterra. Las apuestas se realizan por medio del site de la casa de apuestas en internet. 1.2. Un club ecuestre brinda a sus clientes establos donde pueden guardar los caballos. Este club les ofrece cursos de equitación y paseos. Solamente los socios obtienen acceso a los cursos y a los servicios que se brindan en el establo. Los clientes no socios poseen la posibilidad de participar en paseos y de convertirse en socios. 1.3. El banco de la República brinda diversos servicios a sus clientes con respecto a financiamiento de créditos. El banco ha establecido 3 actividades claramente definidas. Un cliente puede ir al banco y solicitar un crédito. La solicitud de crédito solicitada por un cliente debe de ser sometida a un análisis de viabilidad del crédito. Esto lo realiza el banco para asegurar que el cliente puede soportar el crédito. Posteriormente se puede llegar a firmar el acuerdo final del crédito entre el cliente y el banco. De esta forma se autoriza el crédito para el cliente. 1.4. El propietario de un hotel solicita el desarrollo de un programa para realizar consultas de las habitaciones disponibles y poder realizar las reservaciones de las habitaciones de su hotel. El hotel posee tres tipos de habitaciones: simple, doble y matrimonial. Existen dos tipos de clientes: habituales y ocasionales. Una reservación almacena datos del cliente, de la habitación reservada, la fecha de comienzo y el número de días que será ocupada la habitación. El recepcionista del hotel debe poder hacer las siguientes operaciones: Obtener un listado de las habitaciones disponible de acuerdo a su tipo. Preguntar por el precio de una habitación de acuerdo a su tipo. Preguntar por el descuento ofrecido a los clientes habituales. Preguntar por el precio total para un cliente dado, especificando su número de reserva, tipo de habitación y número de noches. Dibujar en pantalla la foto de una habitación de acuerdo a su tipo. Reservar una habitación especificando el número de la pieza, reserva y nombre del cliente. Eliminar una reserva especificando el número de la habitación. El administrador puede usar el programa para: Cambiar el precio de una habitación de acuerdo a su tipo. Cambiar el valor del descuento ofrecido a los clientes habituales. Calcular las ganancias que tendrán en un mes especificado (considere que todos los meses tienen treinta días).

EJERCICIO SESIÓN NO. 15:	SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN
OBJETIVO:	Ejercitar sesiones 12-14.
CONTENIDO:	Instrucciones: 1. Realizar el modelo de clases para los siguientes enunciados: 1.1. Un club ecuestre brinda a sus clientes establos donde pueden guardar los caballos. Este club les ofrece cursos de equitación y paseos. Solamente los

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.4 ESCUELA CIENCIAS Y SISTEMAS

Figura 6. Páginas 5 – 6 *Hand Book* “Programación intermedios”

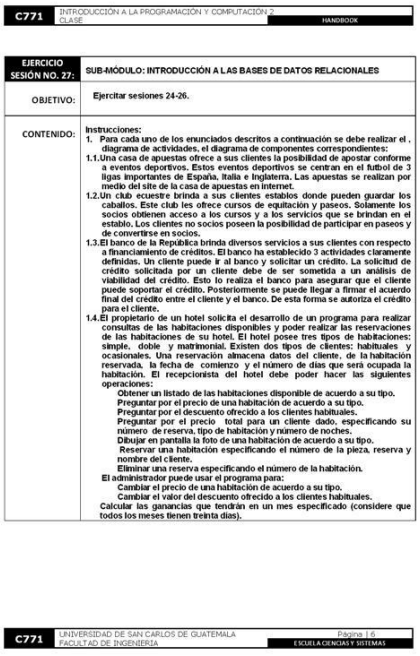


C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2 CLASE HANDBOOK

	socios obtienen acceso a los cursos y a los servicios que se brindan en el establo. Los clientes no socios poseen la posibilidad de participar en paseos y de convertirse en socios. 1.2. El enunciado del problema se encuentra en el archivo con nombre “problema_1_IPC2.txt” dentro de la carpeta “crpDocs” dentro de la carpeta que contiene este Handbook.

EJERCICIO SESIÓN NO. 22:	SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES
OBJETIVO:	Ejercitar sesiones 17-21.
CONTENIDO:	Instrucciones: 1. Para cada uno de los enunciados descritos a continuación se debe realizar el diagrama de secuencia, el diagrama de estados y el diagrama de colaboración correspondientes: 1.1. Una casa de apuestas ofrece a sus clientes la posibilidad de apostar conforme a eventos deportivos. Estos eventos deportivos se centran en el futbol de 3 ligas importantes de España, Italia e Inglaterra. Las apuestas se realizan por medio del site de la casa de apuestas en internet. 1.2. Un club ecuestre brinda a sus clientes establos donde pueden guardar los caballos. Este club les ofrece cursos de equitación y paseos. Solamente los socios obtienen acceso a los cursos y a los servicios que se brindan en el establo. Los clientes no socios poseen la posibilidad de participar en paseos y de convertirse en socios. 1.3. El banco de la República brinda diversos servicios a sus clientes con respecto a financiamiento de créditos. El banco ha establecido 3 actividades claramente definidas. Un cliente puede ir al banco y solicitar un crédito. La solicitud de crédito solicitada por un cliente debe de ser sometida a un análisis de viabilidad del crédito. Esto lo realiza el banco para asegurar que el cliente puede soportar el crédito. Posteriormente se puede llegar a firmar el acuerdo final del crédito entre el cliente y el banco. De esta forma se autoriza el crédito para el cliente. 1.4. El propietario de un hotel solicita el desarrollo de un programa para realizar consultas de las habitaciones disponibles y poder realizar las reservaciones de las habitaciones de su hotel. El hotel posee tres tipos de habitaciones: simple, doble y matrimonial. Existen dos tipos de clientes: habituales y ocasionales. Una reservación almacena datos del cliente, de la habitación reservada, la fecha de comienzo y el número de días que será ocupada la habitación. El recepcionista del hotel debe poder hacer las siguientes operaciones: Obtener un listado de las habitaciones disponible de acuerdo a su tipo. Preguntar por el precio de una habitación de acuerdo a su tipo. Preguntar por el descuento ofrecido a los clientes habituales. Preguntar por el precio total para un cliente dado, especificando su número de reserva, tipo de habitación y número de noches. Dibujar en pantalla la foto de una habitación de acuerdo a su tipo. Reservar una habitación especificando el número de la pieza, reserva y nombre del cliente. Eliminar una reserva especificando el número de la habitación. El administrador puede usar el programa para: Cambiar el precio de una habitación de acuerdo a su tipo. Cambiar el valor del descuento ofrecido a los clientes habituales. Calcular las ganancias que tendrán en un mes especificado (considere que todos los meses tienen treinta días).

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA Página 15 ESCUELA CIENCIAS Y SISTEMAS



C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2 CLASE HANDBOOK

EJERCICIO SESIÓN NO. 27:	SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES
OBJETIVO:	Ejercitar sesiones 24-26.
CONTENIDO:	Instrucciones: 1. Para cada uno de los enunciados descritos a continuación se debe realizar el , diagrama de actividades, el diagrama de componentes correspondientes: 1.1. Una casa de apuestas ofrece a sus clientes la posibilidad de apostar conforme a eventos deportivos. Estos eventos deportivos se centran en el futbol de 3 ligas importantes de España, Italia e Inglaterra. Las apuestas se realizan por medio del site de la casa de apuestas en internet. 1.2. Un club ecuestre brinda a sus clientes establos donde pueden guardar los caballos. Este club les ofrece cursos de equitación y paseos. Solamente los socios obtienen acceso a los cursos y a los servicios que se brindan en el establo. Los clientes no socios poseen la posibilidad de participar en paseos y de convertirse en socios. 1.3. El banco de la República brinda diversos servicios a sus clientes con respecto a financiamiento de créditos. El banco ha establecido 3 actividades claramente definidas. Un cliente puede ir al banco y solicitar un crédito. La solicitud de crédito solicitada por un cliente debe de ser sometida a un análisis de viabilidad del crédito. Esto lo realiza el banco para asegurar que el cliente puede soportar el crédito. Posteriormente se puede llegar a firmar el acuerdo final del crédito entre el cliente y el banco. De esta forma se autoriza el crédito para el cliente. 1.4. El propietario de un hotel solicita el desarrollo de un programa para realizar consultas de las habitaciones disponibles y poder realizar las reservaciones de las habitaciones de su hotel. El hotel posee tres tipos de habitaciones: simple, doble y matrimonial. Existen dos tipos de clientes: habituales y ocasionales. Una reservación almacena datos del cliente, de la habitación reservada, la fecha de comienzo y el número de días que será ocupada la habitación. El recepcionista del hotel debe poder hacer las siguientes operaciones: Obtener un listado de las habitaciones disponible de acuerdo a su tipo. Preguntar por el precio de una habitación de acuerdo a su tipo. Preguntar por el descuento ofrecido a los clientes habituales. Preguntar por el precio total para un cliente dado, especificando su número de reserva, tipo de habitación y número de noches. Dibujar en pantalla la foto de una habitación de acuerdo a su tipo. Reservar una habitación especificando el número de la pieza, reserva y nombre del cliente. Eliminar una reserva especificando el número de la habitación. El administrador puede usar el programa para: Cambiar el precio de una habitación de acuerdo a su tipo. Cambiar el valor del descuento ofrecido a los clientes habituales. Calcular las ganancias que tendrán en un mes especificado (considere que todos los meses tienen treinta días).

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA Página 16 ESCUELA CIENCIAS Y SISTEMAS

C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2 CLASE HANDBOOK

3. PRÁCTICAS PREPARATORIAS

NOMBRE DE CURSO: Introducción a la Programación y Computación 2 CLASE

NOMBRE DEL MODULO: Programación Intermedios (3 prácticas)

PRÁCTICA PREPARATORIA NO. 1:	SUB-MÓDULO: INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES
OBJETIVO:	Ejercitar sesión 1-4.
CONTENIDO:	Instrucciones: Realizar el modelo entidad relación para los siguientes enunciados: 1. El ministerio de comunicaciones, infraestructura y vivienda necesita realizar un sistema que contenga la información correspondiente a municipios, viviendas y personas. El ministerio indica que cada persona solamente puede habitar en una vivienda. La persona puede ser propietaria de varias viviendas. Le interesa especialmente la interrelación de las personas con su cabeza de familia. La capacidad de cada vivienda y cuantas personas habitan en ella. Asimismo quien habita en la vivienda. En el caso de poseer varias viviendas se debe de indicar cual de ellas se reside actualmente. 2. El ministerio de comunicaciones, infraestructura y vivienda solicita diseñar una base de datos que contenga la información correspondiente a todas las carreteras de Guatemala. Se indica que en Guatemala las carreteras se encuentran divididas en tramos. Un tramo siempre pertenece a una única carretera y no puede cambiar de carretera. Un tramo puede pasar por varios términos municipales, siendo un dato de interés el km. del tramo por el que entra en dicho término municipal y el km. por el que sale. Existen una serie de áreas en las que se agrupan los tramos, cada uno de los cuales no puede pertenecer a más de un área.

PRÁCTICA PREPARATORIA NO. 2:	SUB-MÓDULO: ETAPA DE ANALISIS DEL CICLO DE CONSTRUCCIÓN
OBJETIVO:	Ejercitar sesiones 12-14.
CONTENIDO:	Instrucciones: Realizar el diagrama de casos de uso para los siguientes enunciados: 1. Una empresa encargada de vender productos, desea informatizarla y para ello desea que el sistema realice las siguientes funciones: El sistema ha de

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA Página 17 ESCUELA CIENCIAS Y SISTEMAS

C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2 CLASE HANDBOOK

	permitir que los Vendedores introduzcan los productos que venden, junto con sus precios en el sistema. Esta información se empleará para construir listados estadísticos, para que el Director pueda consultarla. Cada mes, se generará un listado especial con agrupaciones de ventas por meses. El sistema, además, ha de permitir al Jefe de Recursos Humanos, dar de alta y borrar a los diferentes vendedores que se añadan o dejen la empresa. En cualquier momento, también tendrá la opción de consultarla. Para la realización de estas tres funcionalidades, se dispondrá de un sistema gestor de base de datos, encargado de traducir las peticiones del Jefe de recursos Humanos. El Director, también podrá consultar en todo momento toda la información referente al personal de la empresa. 2. Se desea realizar un programa en red que simule un juego de Role. El programa estará instalado en un servidor, y tanto el “Master” como los “Jugadores” interaccionan con el sistema, en modo cliente, desde sus casas. El “Master” será el encargado de crear a los personajes, al mundo y el ambiente en el que se desenvolverán los jugadores en el mismo. Eventualmente, el Master se encargará de crear los eventos temporales dentro del mundo, tales como la lluvia, las sequías, ambientes primaverales, etc. El “Master” podrá crear los siguientes tipos de personajes: los guerreros, cuya filosofía es de resolver los problemas por la fuerza; los magos, que mezclan ingenio y magia para resolver las situaciones; y los clérigos, encargados de llevar el poder de los dioses al mundo terrenal. Los “Jugadores”, son los encargados de manejar a sus respectivos personajes, creados previamente por el “Master”, a los cuales les encomendarán misiones y ellos, interaccionando con el mundo, tendrán que resolverlos. Los “Jugadores” podrán manejar a un sólo personaje, y este será o bien un “Guerrero”, un “Mago”, o un “Clérigo”. El “Master”, para dar mayor ambientación al transcurso del juego, también podrá manejar a cualquier de estos tres tipos de personajes, ya que estos interaccionarán con los “Jugadores” para guiarlos, ayudarlos, o entorpecerlos, a lo largo de sus misiones. Al sistema también se podrá conectar todo aquel que desee observar el transcurso de la partida, pero sin poder interaccionar con el sistema, es decir, como meros “Espectadores”.

PRÁCTICA PREPARATORIA NO. 3:	SUB-MÓDULO: ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCION
OBJETIVO:	Ejercitar sesiones 17-22.
CONTENIDO:	Instrucciones: 1. Para cada uno de los enunciados descritos en la Práctica Preparatoria No. 2, se debe realizar el diagrama de clases, el diagrama de secuencia, el diagrama de estados y el diagrama de colaboración correspondientes. 2. A partir del documento “Death by UML.pdf”, que se encuentra dentro de la carpeta “crpDocs” dentro de la carpeta que contiene este Handbook, realizar un resumen del mismo e indicar al finalizar sus opiniones al respecto sobre lo planteado en el mismo.

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA Página 18 ESCUELA CIENCIAS Y SISTEMAS

5. GUÍA DEL INSTRUCTOR DE LA CLASE DE REDES DE NUEVA GENERACIÓN

La guía del instructor del curso de Redes de Nueva Generación, se realizó conforme al formato sugerido en el curso de “Estructuración de cursos y laboratorios”, impartido por el señor Mruntunjaya Panda.

5.1. Datos generales

NOMBRE DE CURSO:	**Redes de Nueva Generación (974)**
PRE- REQUISITOS:	Redes de Computadoras 2 (975)
POST – REQUISITOS:	Ninguno
OBJETIVO:	Al finalizar el curso, el estudiante poseerá la habilidad necesaria para poder utilizar los mecanismos de integración de aplicaciones y tecnologías de Redes de Nueva Generación. Además de comprender los conceptos básicos, fundamentales protocolos y tendencias, que surgen de los servicios en las redes de telecomunicaciones.

MÓDULOS: **Servicios en redes de nueva generación**

VIGENCIA: Primer Semestre 2009

5.2. Distribución del curso

El curso del Redes de Nueva Generación se divide en catorce sesiones, integradas en un módulo denominado Servicios en redes de nueva generación. La tabla XXI muestra la tabla correspondiente a la distribución del curso de NGN.

Tabla XXI. Distribución del curso de NGN

MÓDULO	TEORÍA	PRÁCTICA	TOTAL	EXAMEN	PRÁCTICA PREPARATORIA	PROYECTOS
SERVICIOS EN REDES DE NUEVA GENERACIÓN	22horas	6 horas	28 horas	3	4	1
TOTAL	22horas	6 horas	28horas	3	4	1

5.3. Distribución del módulo

El módulo de servicios en redes de nueva generación se divide en siete sub – módulos. La tabla XXII muestra la tabla correspondiente a la distribución del módulo del curso de NGN.

Tabla XXII. Distribución del módulo de NGN

SUB-MÓDULO	TEORÍA	PRÁCTICA	TOTAL	AL FINALIZAR LOS SUB-MÓDULOS EL ESTUDIANTE SERA CAPAZ DE
SERVICIOS EN REDES FIJAS	4 horas	0 horas	4 horas	Comprender los servicios que pueden ser utilizados en redes fijas.
SERVICIOS EN REDES MÓVILES	4 horas	2 horas	6 horas	Conocer los servicios que pueden ser utilizados en redes móviles.
SERVICIOS EN REDES IP	2 horas	0 horas	2 horas	Comprender los servicios que pueden ser utilizados en redes IP.
REDES DE ALTA VELOCIDAD	2 horas	0 horas	2 horas	Comprender conceptos relacionados a redes de alta velocidad.
CONVERGENCIA DE SERVICIOS	4 horas	2 horas	6 horas	Entender y aplicar la convergencia de servicios.
REDES DE NUEVA GENERACIÓN	4 horas	0 horas	4 horas	Comprender conceptos relacionados a redes de nueva generación.
IP MULTIMEDIA SUBSYSTEM - IMS	2 horas	2 horas	4 horas	Comprender y aplicar el IP *Multimedia Subsystem*.
TOTAL	22 horas	6 horas	28 horas	

5.4. Evaluación

Ponderación de evaluaciones, tareas, prácticas y laboratorio. La tabla XXIII muestra la tabla correspondiente a la ponderación de evaluación del curso de NGN.

Tabla XXIII. Ponderación de la evaluación de NGN

ACTIVIDAD	NÚMERO	PORCENTAJE	TOTAL
PRÁCTICA PREPARATORIA	4	0.75 %	3.00 %
AUTO-EVALUACIÓN	2	1.00 %	2.00 %
LABORATORIO	1	40.0 %	40.00 %
EXAMEN PARCIAL	2	15.0 %	30.00 %
EXAMEN FINAL	1	25.0 %	25.00 %
		TOTAL	**100.00 %**

5.5. Detalle de sesiones

La tabla XXIV muestra la tabla correspondiente al detalle de sesiones del curso de NGN.

Tabla XXIV. Detalle de sesiones de NGN

NO	TEORÍA	NO	PRÁCTICA	PRÁCTICA PREPARATORIA / PROYECTO
1	**SERVICIOS EN REDES DE NUEVA GENERACION – 1**			
2	**SERVICIOS EN REDES DE NUEVA GENERACION – 2**			
3	**SERVICIOS EN REDES DE NUEVA GENERACION – 3**			
4	**SERVICIOS EN REDES DE NUEVA GENERACION – 4**	5	**SERVICIOS EN REDES DE NUEVA GENERACION – 5**	
6	**SERVICIOS EN REDES DE NUEVA GENERACION – 6**			
7	**SERVICIOS EN REDES DE NUEVA GENERACION – 7**			Práctica No.1
8	**SERVICIOS EN REDES DE NUEVA GENERACION – 8**			
9	**SERVICIOS EN REDES DE NUEVA GENERACION – 9**	10	**SERVICIOS EN REDES DE NUEVA GENERACION – 10**	Práctica No.2
11	**SERVICIOS EN REDES DE**			

	NUEVA GENERACION – 11			
12	**SERVICIOS EN REDES DE NUEVA GENERACION – 12**			Práctica No.3
13	**SERVICIOS EN REDES DE NUEVA GENERACION – 13**	14	**SERVICIOS EN REDES DE NUEVA GENERACION – 14**	Práctica No.4

5.6. Distribución de sesiones

La tabla XXV muestra la tabla correspondiente a la distribución de sesiones del curso de NGN.

Tabla XXV. Distribución de sesiones de NGN

SESIÓN NO. 1:	**SUB-MÓDULO: SERVICIOS EN REDES FIJAS**
OBJETIVO:	**Introducir a los estudiantes sobre los conceptos básicos de redes inteligentes.**
CONTENIDO:	**1. Red Inteligente** **1.1. Conceptos** **1.1.1. Sistema de Red Inteligente** **1.1.1.1. SSP (*service switching point*)** **1.1.1.2. SCP (*service control point*)** **1.1.1.3. SMS (*service management system*)** **1.1.2. Características de una Red Inteligente** **1.1.3. Servicios** **1.1.3.1. CS1** **1.1.3.2. CS2**

	1.1.3.3. CS3 **1.2. Contexto** **1.3. Evolución** **1.4. Componentes AIN (*Advanced Intelligent Network*)** **1.4.1. CCS (*Common Channel Signaling*)** **1.4.2. AIN (*Advanced Intelligent Network*)** **1.4.2.1. Nodos** **1.4.2.2. Elementos**
SESIÓN NO. 2:	**SUB-MÓDULO: SERVICIOS EN REDES FIJAS**
OBJETIVO:	**Presentar a los estudiantes las arquitecturas y servicios de redes inteligentes.**
CONTENIDO:	**1. Redes Inteligentes** **1.1. Arquitectura Funcional** **1.1.1. Modelo Conceptual** **1.1.1.1. Planos** **1.1.2. Fase de Especificación de Servicios (CS1)** **1.1.2.1. Entidades** **1.1.2.2. Diagrama** **1.2. Modelo de Llamada Básico** **1.2.1. PIC** ***1.2.2. Trigger*** **1.2.3. Servicio AIN** **1.2.4. Puntos en Llamada** **1.3. Dimensionado de un SCP (*Service Control Point*)** **1.3.1. Diseño para Dimensionar un Nodo SCP**
SESIÓN NO. 3:	**SUB-MÓDULO: SERVICIOS EN REDES MÓVILES**
OBJETIVO:	**Brindar a los estudiantes los conceptos sobre redes móviles.**

CONTENIDO:	**1. Redes Móviles de primera generación.** **1.1. AMP** **1.1.1. Características** **1.1.2. Canales** **1.1.2.1. FOCC (*forward control channel*)** **1.1.2.2. RECC (*reverse control channel*)** **1.1.2.3. FVC (*forward voice channel*)** **1.1.2.4. RVC (*reverse voice channel*)** **1.1.3. Identificadores** **1.1.3.1. ESN (*electronic serial number*)** **1.1.3.2. SID (*system identification*)** **1.1.3.3. MIN (*moble identification number*)** **1.2. D-AMP** **1.2.1. Propósito** **1.2.2. Canales** **1.2.2.1. FOCC (*forward control channel*)** **1.2.2.2. RECC (*reverse control channel*)** **1.2.2.3. FVC (*forward voice channel*)** **1.2.2.4. RVC (*reverse voice channel*)** **1.2.2.5. FDTC (*reverse analog control channel*)** **1.2.2.6. RDTC (*reverse digital traffic channel*)** **2. Redes Móviles de segunda generación – 2G.** **2.1. GSM (*Global System for Mobile Communication*)** **2.1.1. Topología de GSM** **2.1.2. Interfaces** **2.1.2.1. Um** **2.1.2.2. Abis** **2.1.2.3. A** **2.1.2.4. MAP** **2.1.3. SIM (*Subscriber Identity Module*)** **2.1.4. MAP (*Mobile Application Part*)** **2.2. CDMA IS-95** ***2.2.1. Spread Spectrum*** **2.2.1.1. *Frequency Hopping Spread* (FHSS)** ***2.2.1.2. Direct Sequence Spread Spectrum*** **2.2.2. Arquitectura** ***2.2.2.1. Base Station*** ***2.2.2.2. Mobile Station***

	2.2.2.3. ***Mobile Switching Center*** **(MSC)** **2.2.2.4.** ***Home Location Register*** **(HLR)** **2.2.2.5.** ***Vistied Location Register*** **(VLR)** **2.2.2.6.** ***Authentication Center*** **(AC)** ***2.2.2.7.*** ***Operations Support, Billin Systems, Interworking Function*** **2.2.3. Canales** **2.2.3.1. Físicos** **2.2.3.2. Lógicos** ***2.2.3.2.1. Pilot Channel*** ***2.2.3.2.2. Paging Channel*** ***2.2.3.2.3. Sync Channel*** ***2.2.3.2.4. Access Channel*** ***2.2.3.2.5. Forward Traffic Channels*** ***2.2.3.2.6. Reverse Traffic Channels*** **2.2.4. Interfaces** **2.2.4.1. A Interface (BSC-MSC)** **2.2.4.2. Abis Interface (BTS-BSC)** **2.2.4.3. B Interface (MSC-VLR)** **2.2.4.4. C Interface (MSC-HLR)** **2.2.4.5. D Interface (HLR-VLR)** **2.2.4.6. E Interface (MSC-MSC)** **2.2.4.7. H Interface (HLR-AC)** **2.2.4.8. L Interface (MSC-IWF)** **2.2.4.9. Um Interface (BS-MS)** **2.2.5. IS-41** **3. Red Inteligente Inalámbrica (WIN)** **3.1. Características** **3.2. Ventajas** **3.3. Diferencia entre Servicios** **3.3.1. Servicios de Movilidad Terminal** **3.3.2. Servicios de Movilidad Personal** **3.3.3. Servicios de Redes Avanzadas** **3.4. Administración del Servicio** **4. Generación** **4.1. 2.5 G** **4.2. 3 G** **4.3. 4 G**

SESIÓN NO. 4:	**SUB-MÓDULO: SERVICIOS EN REDES MÓVILES**
OBJETIVO:	**Introducir a los estudiantes conceptos sobre la arquitectura de servicios en redes móviles.**
CONTENIDO:	**1. Arquitecturas de Servicios.** **1.1. CAMEL.** **1.1.1. Antecedentes** **1.1.2. Arquitectura** **1.1.3. CAMEL Control** **1.2. WIN.** **1.2.1. Antecedentes** **1.2.2. Servicios** ***1.2.2.1. Hands-Free, Voice-Controlled Service*** **1.2.2.2. Data Capabilities** **1.2.3. Componentes** **2. Tecnología 3G.** **2.1. Antecedentes** **2.2. Conceptos** **2.3. Interfaces** **2.3.1. WCDMA EDGE** **2.3.2. 1XRTT o cdma2000** **2.4. UMTS (*Universal Mobile Telephony System*)** ***2.4.1. Services Layer*** ***2.4.2. Control Layer*** ***2.4.3. Connectivity Layer***
SESIÓN NO. 5:	**SUB-MÓDULO: SERVICIOS EN REDES MÓVILES**
OBJETIVO:	**Evaluar los contenidos de las sesiones 1- 4 aprendidos por los estudiantes.**
	Examen Parcial número 1.

CONTENIDO:	
SESIÓN NO. 6:	**SUB-MÓDULO: SERVICIOS EN REDES IP**
OBJETIVO:	**Introducir a los estudiantes los servicios utilizados en redes IP.**
CONTENIDO:	**1. Redes IP y Multimedia** **1.1. Redes IP** **1.2. Evolución de los Sistemas de TV** **1.2.1. Televisión Analógica. Formato y Estándares** **1.2.2. Televisión Digital. Formato y Estándares** **1.2.3. *Mobile* TV** **1.2.4. IPTV** **1.3. Servicios Avanzados** ***1.3.1. Triple-Play*** ***1.3.2. Quad-Play*** ***1.3.3. Triple-Play vs Quad-Play*** **1.4. Infraestructura y Protocolos Necesarios** **1.4.1. Infraestructura Necesaria** **1.4.2. Modelo de Capas de Comunicación**
SESIÓN NO. 7:	**SUB-MÓDULO: REDES DE ALTA VELOCIDAD**
OBJETIVO:	**Presentar los conceptos fundamentales respecto a redes de alta velocidad.**
CONTENIDO:	***1. Fast Ethernet*** **1.1. Historia** **1.2. Diferencias entre 100BaseT y 10BaseT** **1.3. Cableado** **1.4. Tipos**

	1.4.1. 100 Base T **1.4.1.1. Transmisión** **1.4.1.2. Especificaciones** **1.4.1.3. Tiempos de Respuesta** **1.4.2. 100 VG- AnyLAN** **1.4.2.1. Transmisión** ***2. Gigabit Ethernet*** **2.1. Subcapas** **2.1.1. 10GBASE-X** **2.1.2. 10GBASE-R** **2.1.3. 10GBASEW** **2.1.4. XGMII** **2.1.5. XAUI** **2.1.6. XSBI** **2.2. Arquitectura** **2.3. Trama** **3. FDDI** **3.1. FDDI *Operation*** **3.2. Arquitectura FDDI** **3.3. Topología FDDI** **4. ATM** **4.1. Formato Básico de la Celda ATM** ***4.2. Protocol Reference Model*** **4.3. Capa ATM** ***4.3.1. Virtual Path Level*** ***4.3.2. Virtual Channel Level***
SESIÓN NO. 8:	**SUB-MÓDULO: CONVERGENCIA DE SERVICIOS**
OBJETIVO:	**Entender la convergencia de servicios de redes fijas, redes móviles y redes ip.**
CONTENIDO:	**1. SPIRITS (*Services in the PSTN Requesting Internet Services*).** **1.1. Jerarquía** **1.2. Arquitectura**

	2. PINT (PSTN/*Internet Internetworking*). **2.1. Servicios Básicos** **2.2. Interacciones Cliente/Servidor** **2.3. Protocolos** **2.3.1. SIP** **2.3.2. SDP** **2.4. Arquitectura** **3. OSA/Parlay.** **3.1. Servicios** **3.2. OSA/Parlay *Gateway*** **3.2.1. Elementos** **3.2.1.1. FW** **3.2.1.2. SCFs** **3.2.2. *Detailed* OSA *Framework Modeling*** **3.3. Parlay X.** **3.3.1. Características**
SESIÓN NO. 9:	**SUB-MÓDULO: CONVERGENCIA DE SERVICIOS**
OBJETIVO:	**Aprender la importancia de la integración de la telefonía y la computación.**
CONTENIDO:	**1. Integración de la Telefonía y la Computación (CTI).** **1.1. ASR (*Automatic Speech Recognition*).** **1.1.1. *Human-Computer Interaction* ASR** **1.1.2. Algoritmo** **1.1.3. Acceso Remoto** **1.1.4. Aplicaciones** **1.2. TTS (*Text to Speech*).** **1.2.1. Características** **1.2.2. Retos** **1.2.3. Internet** **1.3. Sistemas IVR.** **1.3.1. Proceso** **1.3.2. Beneficios**

	1.3.3. Tecnologías **1.3.3.1. TTS** **1.3.3.2. DTMF** **1.4. *Voice* XML.** **1.4.1. Estándar** **1.4.2. Aplicaciones** **1.4.3. Diálogos y Subdiálogos** **1.4.4. Gramáticas** **1.4.5. Eventos** **1.4.6. Enlaces** **1.5. Aplicaciones de Telefonía IP con software libre (*Asterisk*).** **1.5.1. Servicios** **1.5.2. Soporte** **1.5.3. Versiones** **1.5.4. Compatibilidad**
SESIÓN NO. 10:	**SUB-MÓDULO: CONVERGENCIA DE SERVICIOS**
OBJETIVO:	**Evaluar los contenidos de las sesiones 6- 9 aprendidos por los estudiantes.**
CONTENIDO:	**Examen Parcial número 2.**
SESIÓN NO. 11:	**SUB-MÓDULO: REDES DE NUEVA GENERACIÓN**
OBJETIVO:	**Introducir a los estudiantes el concepto de redes de nueva generación.**
CONTENIDO:	**1. NGN Visión, Escenarios y Avances** **1.1. Perspectivas y Potenciales** **1.2. Posibles Escenarios** **1.3. Avances** **2. NGN Requerimientos en Tecnología y Administración** **2.1. Requerimientos en Tecnología**

	2.2. Requerimientos en Administración **3. Arquitectura Funcional** **3.1. Arquitectura ITU** **3.2. Arquitectura Sugerida**
SESIÓN NO. 12:	**SUB-MÓDULO: REDES DE NUEVA GENERACIÓN**
OBJETIVO:	**Entender los aspectos concernientes a la migración hacia una NGN, y los aspectos económicos y regulatorios a considerar.**
CONTENIDO:	**1. NGN Operador, Proveedor, Cliente y CTE.** **1.1.Operador de Red** **1.1.1. Tipos Fragmentados de Operadores NGN** **1.1.1.1. Operador de Transporte** **1.1.1.2. Operador de Acceso** **1.1.1.3. Operador del Núcleo** **1.2. Proveedor de Servicios** **1.2.1. Tipos de Proveedores** **1.2.1.1. Proveedores de Servicios** **1.2.1.2. Proveedores de Aplicaciones** **1.2.1.3. Proveedores de Contenido** **1.2.1.4. Proveedores de Información** **1.3. Cliente y CTE** **2. Evolución de la red y servicios hacia NGN.** **2.1. Pasos de la Mayor Evolución para las Redes y Servicios de Hoy** **2.2. Evolución de las Redes Fijas** **2.3. Evolución de las Redes Móviles** **2.4. Evolución de las Redes de Cable** **2.5. Evolución del Internet** **2.6. Problemas Críticos de Redes IP a Resolver** **3. NGN Desarrollo Áreas Claves.** ***3.1. Terminal Area*** ***3.2. Access Network Area*** ***3.3. Backhaul Network Area***

	3.4. Core Transport Network Area ***3.5. Service Creation Area*** ***3.6. Network Control and Management Area*** ***3.7. Service Control and Management*** ***3.8. Advanced Technologies for Network and Service Management*** ***3.8.1. Intelligent Agent*** ***3.8.2. Artificial Intelligence*** **3.8.3. SON** **3.8.4. GRID**
SESIÓN NO. 13:	**SUB-MÓDULO: IP MULTIMEDIA SUBSYSTEM – IMS**
OBJETIVO:	**Explicar al estudiante en qué consiste un IMS**
CONTENIDO:	**1. Características de IMS (*IP Multimedia Subsystem*).** **2. Arquitectura de servicios IMS.** **2.1. Capa de Servicio** **2.2. Capa de Control** **2.3. Capa de Transporte** **2.4. Capa de Dispositivo** **3. Aplicaciones de Nueva Generación basadas en IMS.** **3.1. SIP (*Session Initiation Protocol*)** **3.2. JAIN (*Java Advanced Intelligent Network*)**
SESIÓN NO. 14:	**SUB-MÓDULO: IP MULTIMEDIA SUBSYSTEM – IMS**
OBJETIVO:	**Evaluar los contenidos de las sesiones 1- 13 aprendidos por los estudiantes.**
CONTENIDO:	**Examen Final.**

5.7. Detalle de conducción de tutoriales

La tabla XXVI muestra la tabla correspondiente al detalle de conducción de tutoriales del curso de NGN.

Tabla XXVI. Detalle de conducción de tutoriales de NGN

ACTIVIDAD	DETALLE
Iniciado en sesión de clase número:	N.A.
Finalizado en sesión de clase número	N.A.
Viva / Demo para ser conducido:	N.A.
Criterio de evaluación:	N.A.
Viva:	N.A.
Documentación:	N.A.

5.8. Detalle de conducción de exámenes

La tabla XXVII muestra la tabla correspondiente al detalle de conducción de exámenes del curso de NGN.

Tabla XXVII. Detalle de conducción de exámenes de NGN

EXAMEN NÚMERO:	**1**
Examen a ser realizado después de la sesión número:	4
Temas para el examen:	Sesión 1 – Sesión 4
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%

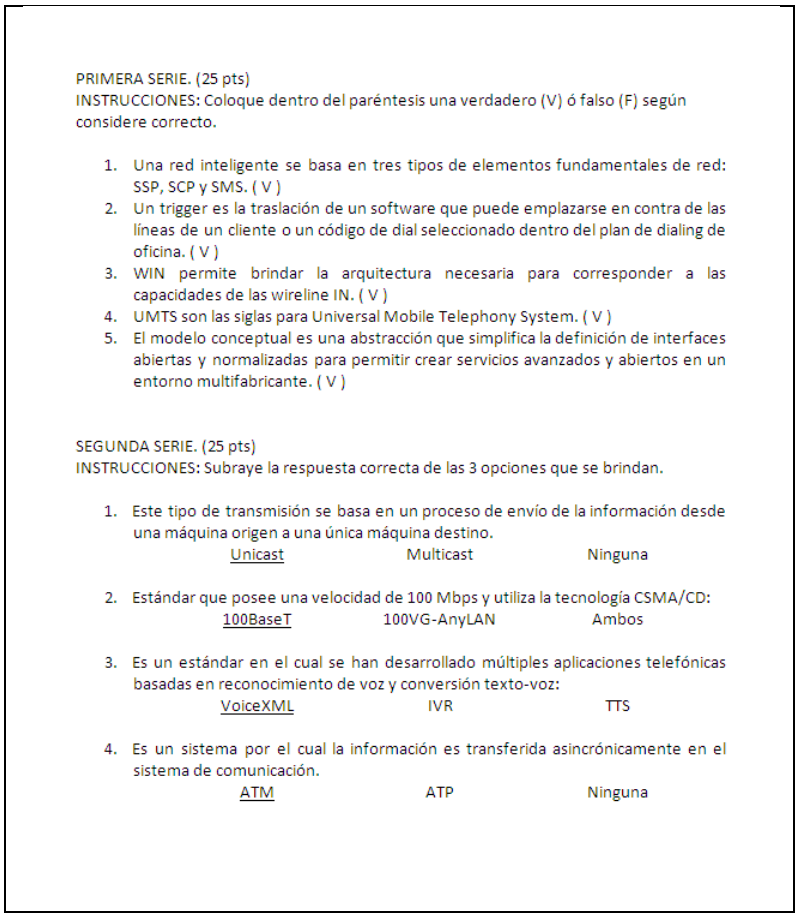
EXAMEN NÚMERO:	**2**
Examen a ser realizado después de la sesión número:	9
Temas para el examen:	Sesión 6 – Sesión 9
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%

EXAMEN NÚMERO:	**3**
Examen a ser realizado después de la sesión número:	13
Temas para el examen:	Sesión 1 – 13, haciendo especial énfasis en las sesiones 11 – 13
Duración del Examen:	2 horas
Distribución del Examen:	Objetivo – 100%

5.9. Ejemplo de examen

La figura 7 muestra la figura correspondiente al examen final de ejemplo del curso de NGN.

Figura 7. Examen final de ejemplo de NGN

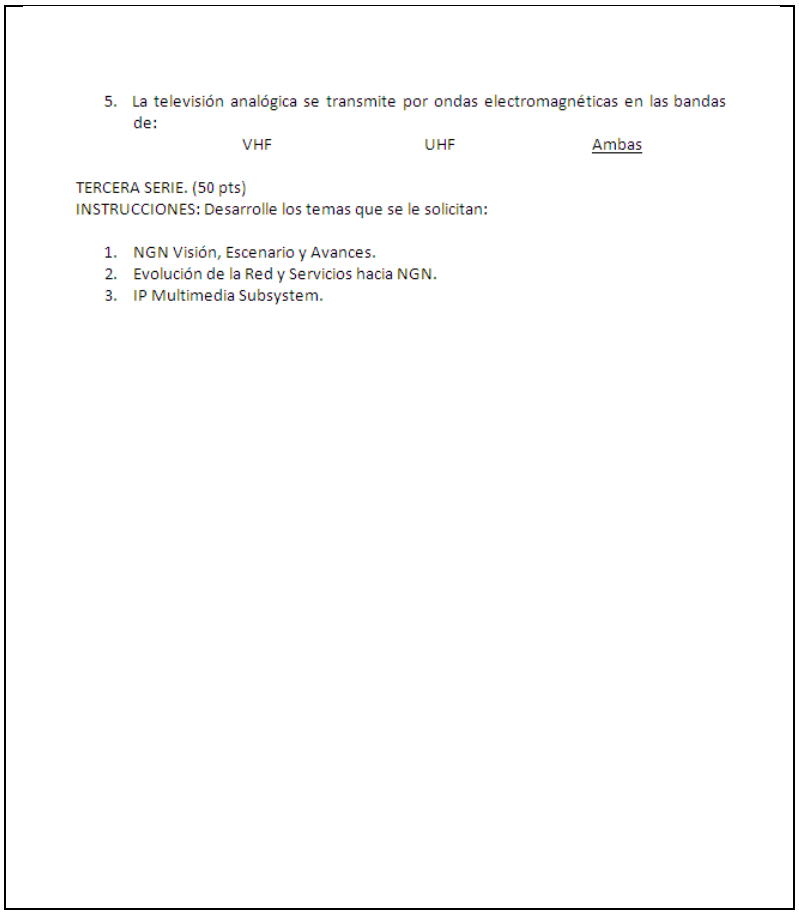


PRIMERA SERIE. (25 pts)
INSTRUCCIONES: Coloque dentro del paréntesis una verdadero (V) ó falso (F) según considere correcto.

1. Una red inteligente se basa en tres tipos de elementos fundamentales de red: SSP, SCP y SMS. (V)
2. Un trigger es la traslación de un software que puede emplazarse en contra de las líneas de un cliente o un código de dial seleccionado dentro del plan de dialing de oficina. (V)
3. WIN permite brindar la arquitectura necesaria para corresponder a las capacidades de las wireline IN. (V)
4. UMTS son las siglas para Universal Mobile Telephony System. (V)
5. El modelo conceptual es una abstracción que simplifica la definición de interfaces abiertas y normalizadas para permitir crear servicios avanzados y abiertos en un entorno multifabricante. (V)

SEGUNDA SERIE. (25 pts)
INSTRUCCIONES: Subraye la respuesta correcta de las 3 opciones que se brindan.

1. Este tipo de transmisión se basa en un proceso de envío de la información desde una máquina origen a una única máquina destino.
   Unicast Multicast Ninguna
2. Estándar que posee una velocidad de 100 Mbps y utiliza la tecnología CSMA/CD:
   100BaseT 100VG-AnyLAN Ambos
3. Es un estándar en el cual se han desarrollado múltiples aplicaciones telefónicas basadas en reconocimiento de voz y conversión texto-voz:
   VoiceXML IVR TTS
4. Es un sistema por el cual la información es transferida asincrónicamente en el sistema de comunicación.
   ATM ATP Ninguna



5. La televisión analógica se transmite por ondas electromagnéticas en las bandas de:

VHF UHF Ambas

TERCERA SERIE. (50 pts)
INSTRUCCIONES: Desarrolle los temas que se le solicitan:

1. NGN Visión, Escenario y Avances.
2. Evolución de la Red y Servicios hacia NGN.
3. IP Multimedia Subsystem.

5.10. Bibliografía recomendada

La tabla XXVIII muestra la tabla correspondiente a la bibliografía recomendada del curso de NGN.

Tabla XXVIII. Bibliografía recomendada de NGN

No.	Libro
1	Wilkinson Neill. ***Next Generation Network Services: Technologies & Strategies*.** Editorial *Prentice Hall.* 1a Edición. México. Año 2003.
2	Anderson John R. ***Intelligent Networks: Principles and Applications (IEE Telecommunications Series, 46)***. Editorial *McGraw-Hill* / Interamericana de España, S. A. 2ª Edición. España. Año 2002.
3	Zuidweg Johan. ***Next Generation Intelligent Networks (Artech House Telecommunications Library).*** Editorial *Artech House*. 1a Edición. Reino Unido. Año 2002.
4	Poikselka Miikka, Niemi Aki, Khartabil Hisham, Mayer Georg. ***The IMS: IP Multimedia Concepts and Services.*** Editorial *John Wiley & Sons*. 2ª Edición. EUA. Año 2006.
5	Camarillo Gonzalo, García Martín Miguel Angel. ***The 3G IP Multimedia Subsystem(IMS): Merging the Internet and the Cellular Worlds.*** Editorial *John Wiley & Sons*. 2ª Edición. EUA. Año 2006.
6	Noldus Rogier. ***CAMEL: Intelligent Networks for the GSM, GPRS and UMTS Network.*** Editorial *John Wiley & Sons.* 2ª Edición. EUA. Año 2006.
7	Alan B. Johnston. ***SIP: Understanding the Session Initiation Protocol.*** Editorial *Artech House*. 2a Edición. Reino Unido. Año 2004.
8	Henry Sinnreich, Alan Johnson and Robert Sparks. ***SIP Beyond VoIP. IP TELEPHONY COOKBOOK*** *(http://www.terena.nl/library/IPTELEPHONYCOOKBOOK/).* Año 2005.
9	Antoni Barba Martí, Joaquim Casal i Fàbrega, Antoni Barba Martí. **Gestión de Red**. Ediciones UPC. 1a Edición. Universidad Politécnica de Cataluña. Año 1999.
10	Jaime Lloret Mauri, Miguel García Pineda, Fernando Boronat Seguí. **IPTV: la television por internet.** Publicaciones Vértice. 1a Edición. España. Año 2009.
11	*John Wiley and Sons.* ***Next Generation Networks: perspectives and potentials****.* Editorial *John Wiley & Sons*. 2ª Edición. EUA. Año 2008.
12	Nathan J. Muller. ***Desktop encyclopedia of telecommunications.*** Editorial *McGraw-Hill Professional.* 1a Edición. EUA. Año 2002.
13	Klaus Dembowski, Virginia Pérez Moreno, José Luis Cortés, Nuria González. **Gran libro hardware: Información sobre la totalidad del hardware, de rápido acceso.** Editorial Marombo,

1a Edición. España. Año 2003.

14 Antonio Salavert Casamor. **Los protocolos en las redes de ordenadores.** Editorial *Pearson* Educación, 1a Edición. España. Año 2003.

15 Laurie G. Cuthbert, Jean-Claude Sapanel. ***ATM: The Broadband Telecommunications Solution****. IEE telecommunications series*. 1a Edición. Londres. Año 1993.

16 María Carmen España Boquera. **Servicios avanzados de telecomunicación.** Ediciones Díaz de Santos. 1a Edición. España. Año 2003.

17 Stephen M. Mueller. ***APIs and Protocols for Convergent Network Services.*** Editorial *McGraw-Hill Professional.* 1a Edición. EUA. Año 2002.

18 Vijay K. Gurbani, Xian-He Sun. ***Architecting the telecommunication evolution: toward converged network services.*** *CRC Press.* 1a Edición. EUA. Año 2006.

19 Heikki Kaaranen, Ari Ahtiainen, Lauri Laitinen, Siamäk Naghian, Valtteri Niemi. ***UMTS Networks: Architecture, Mobility, and Services.*** Editorial *John Wiley & Sons.* 2ª Edición. EUA. Año 2005.

20 Maria Antònia Martí Antonín, Joaquim Llisterri Boix. **Tecnologías del texto y del habla.** Ediciones Universitarias 1a Edición. Barcelona. Año 2004.

21 Antoni Barba Marti, Xavier Hesselbach Serra, Antoni Barba Martí. **Inteligencia de red.** Ediciones UPC. 1a Edición. España. Año 2002.

22 Jan P. H. Van Santen, J. Olive, Julia Hirschberg. ***Progress in speech synthesis.*** Editorial *Springer*. 1a Edición. EUA. Año 1996.

23 Zheng-Hua Tan, Børge Lindberg. ***Automatic Speech Recognition on Mobile Devices and Over Communication Networks.*** Editorial *Springer*. 1a Edición. EUA. Año 2008.

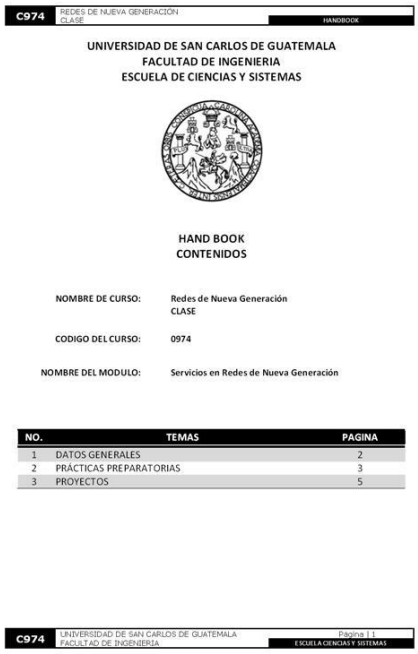
24 Julie A. Jacko, Andrew Sears. ***The human-computer interaction handbook: fundamentals, evolving technologies, and emerging applications.*** Editorial *Lawrence Erlbaum Associates.* 1a Edición. EUA. Año 2003.

25 Freddy Ghys, Marcel Mampaey, Michel Smouts, Arto Vaaraniemi. ***3G multimedia network services, accounting, and user profiles*****.** Editorial *Artech House.* 1a Edición. Reino Unido. Año 2003.

5.11. Hand Book

El *Hand Book* de NGN contiene los ejercicios y prácticas que son citadas en el IG. El *Hand Book* se desplegará en forma de imágenes. Éste contiene tres secciones: Ejercicios, Prácticas Preparatorias y Proyectos.

Figura 8. Páginas 1 – 4 *Hand Book* “Servicios en redes de nueva generación”



C974 REDES DE NUEVA GENERACIÓN CLASE HANDBOOK

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE CIENCIAS Y SISTEMAS

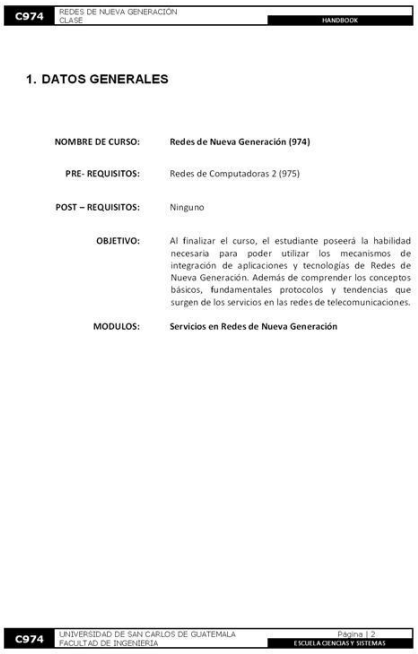
HAND BOOK
CONTENIDOS

NOMBRE DE CURSO: Redes de Nueva Generación CLASE

CODIGO DEL CURSO: 0974

NOMBRE DEL MODULO: Servicios en Redes de Nueva Generación

C974 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.1 ESCUELA CIENCIAS Y SISTEMAS



C974 REDES DE NUEVA GENERACIÓN CLASE HANDBOOK

1. DATOS GENERALES

NOMBRE DE CURSO: Redes de Nueva Generación (974)

PRE- REQUISITOS: Redes de Computadoras 2 (975)

POST – REQUISITOS: Ninguno

OBJETIVO: Al finalizar el curso, el estudiante poseerá la habilidad necesaria para poder utilizar los mecanismos de integración de aplicaciones y tecnologías de Redes de Nueva Generación. Además de comprender los conceptos básicos, fundamentales protocolos y tendencias que surgen de los servicios en las redes de telecomunicaciones.

MODULOS: Servicios en Redes de Nueva Generación

C974 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.2 ESCUELA CIENCIAS Y SISTEMAS

C974 REDES DE NUEVA GENERACIÓN CLASE HANDBOOK

2. PRÁCTICAS PREPARATORIAS

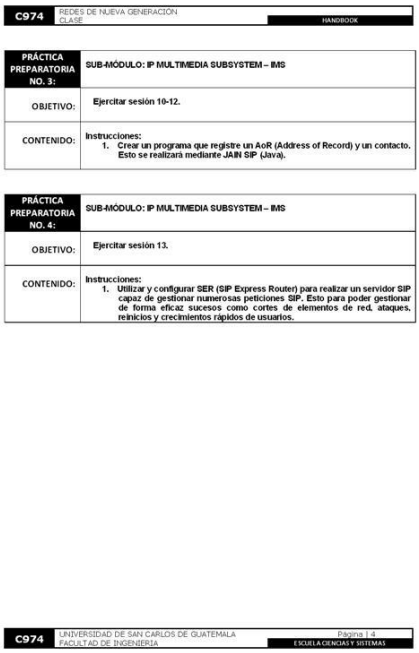
NOMBRE DE CURSO: Redes de Nueva Generación CLASE

NOMBRE DEL MODULO: Servicios en Redes de Nueva Generación (4 prácticas)

PRÁCTICA PREPARATORIA NO. 1:	SUB-MÓDULO: REDES DE ALTA VELOCIDAD
OBJETIVO:	Ejercitar sesiones 1-7.
CONTENIDO:	Instrucciones: 1. Utilizar un software simulador/emulador NGN para: 1.1. Realizar una emulación de latencia en redes Gigabit Ethernet. 1.2. Realizar una simulación de escenarios críticos para redes de almacenamiento (SAN). 1.3. Realizar un análisis de latencia en transporte de video y VoIP. 1.4. Realizar una emulación de degradación selectiva producida por routers y switches NGN (cambios de direcciones, pérdidas de ciertas tramas, SLVANs, MPLS). 1.5. Realizar una emulación de retardo doppler generado, presentes en WAN satelitales. • El software que se recomienda para realizar la práctica es: o NCTUns (National Chiao Tung University, Network Simulator). (software libre con distribución de código abierto) o OMNET ++. (software gratuito para propósitos académicos) o OPNET MODELER. (software propietario)

PRÁCTICA PREPARATORIA NO. 2:	SUB-MÓDULO: CONVERGENCIA DE SERVICIOS
OBJETIVO:	Ejercitar sesiones 8-9.
CONTENIDO:	Instrucciones: 1. Instalar y configurar Asterisk (la versión estable actual) con todas sus funcionalidades para implementar una central avanzada de telefonía. Esto para que se pueda realizar una central a la que puedan accesar otras PC.

C974 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.3 ESCUELA CIENCIAS Y SISTEMAS



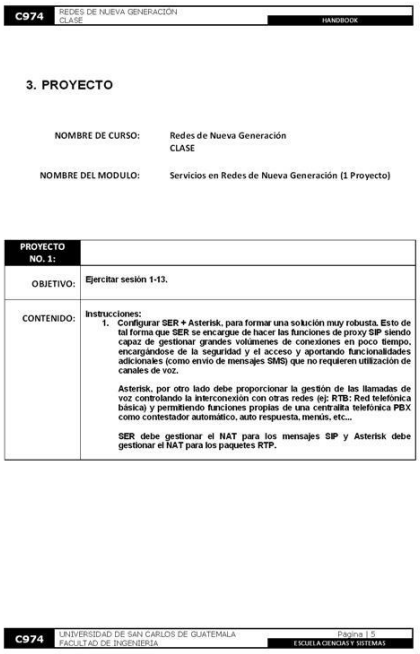
C974 REDES DE NUEVA GENERACIÓN CLASE HANDBOOK

PRÁCTICA PREPARATORIA NO. 3:	SUB-MÓDULO: IP MULTIMEDIA SUBSYSTEM – IMS
OBJETIVO:	Ejercitar sesión 10-12.
CONTENIDO:	Instrucciones: 1. Crear un programa que registre un AoR (Address of Record) y un contacto. Esto se realizará mediante JAIN SIP (Java).

PRÁCTICA PREPARATORIA NO. 4:	SUB-MÓDULO: IP MULTIMEDIA SUBSYSTEM – IMS
OBJETIVO:	Ejercitar sesión 13.
CONTENIDO:	Instrucciones: 1. Utilizar y configurar SER (SIP Express Router) para realizar un servidor SIP capaz de gestionar numerosas peticiones SIP. Esto para poder gestionar de forma eficaz sucesos como cortes de elementos de red, ataques, reinicios y crecimientos rápidos de usuarios.

C974 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERIA Página 1.4 ESCUELA CIENCIAS Y SISTEMAS

Figura 9. Página 5 *Hand Book* “Servicios en redes de nueva generación”



C974 REDES DE NUEVA GENERACIÓN CLASE HANDBOOK

3. PROYECTO

NOMBRE DE CURSO:	Redes de Nueva Generación CLASE
NOMBRE DEL MODULO:	Servicios en Redes de Nueva Generación (1 Proyecto)

PROYECTO NO. 1:	
OBJETIVO:	Ejercitar sesión 1-13.
CONTENIDO:	Instrucciones: 1. Configurar SER + Asterisk, para formar una solución muy robusta. Esto de tal forma que SER se encargue de hacer las funciones de proxy SIP siendo capaz de gestionar grandes volúmenes de conexiones en poco tiempo, encargándose de la seguridad y el acceso y aportando funcionalidades adicionales (como envío de mensajes SMS) que no requieren utilización de canales de voz. Asterisk, por otro lado debe proporcionar la gestión de las llamadas de voz controlando la interconexión con otras redes (ej: RTB: Red telefónica básica) y permitiendo funciones propias de una centralita telefónica PBX como contestador automático, auto respuesta, menús, etc... SER debe gestionar el NAT para los mensajes SIP y Asterisk debe gestionar el NAT para los paquetes RTP.

C974 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA Página 1.6 ESCUELA CIENCIAS Y SISTEMAS

6. DOCUMENTACIÓN DE APOYO DEL CURSO DE INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1

6.1. Descripción

Documentación de apoyo del curso de IPC1, el cual fue realizado en formato de un libro que contiene las unidades y temas más importantes impartidos en el curso.

El libro consta de siete capítulos. Estos capítulos son:

a. Introducción a la computación y programación.
b. Programación orientada a objetos.
c. Introducción a la plataforma java.
d. Elementos básicos de programación y programación estructurada.
e. Estructuras algorítmicas.
f. Flujo (*Streams*) y manipulación de archivos.
g. Tipo de datos abstractos.

Figura 10. Páginas 1 – 4 Libro “Programación principiantes”

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

ESCUELA DE CIENCIAS Y SISTEMAS

LIBRO

NOMBRE DE CURSO: Introducción a la Programación y Computación 1

CODIGO DEL CURSO: 0770

NOMBRE DEL MODULO: Programación Principiantes

AUTOR: Mario José Bautista Fuentes

CONTENIDO

1. INTRODUCCIÓN A LA COMPUTACIÓN Y PROGRAMACIÓN

1.1. Conceptos Computacionales

1.1.1. Computadora

La computadora es un dispositivo capaz de aceptar información. Esta información se encuentra en forma de datos digitalizados.

La computadora es capaz de manipular los datos como resultado de un programa o una secuencia de instrucciones. Estas instrucciones indican como los datos deben ser procesados.

1.1.2. Hardware

El hardware es la parte física. Este se encuentra integrado por los componentes físicos de la computadora.

El hardware es el que le indica a los componentes físicos que deben hacer. El hardware comprende los componentes electrónicos, tarjetas, periféricos y equipo.

1.1.3. Software

El software son los programas que se ejecutan en la computadora y los datos almacenados en la misma.

El software puede ser de dos tipos:

- Software de sistema: este son programas cuyo fin permiten operar la computadora.
- Software de aplicación: este son programas que permiten al usuario realizar alguna tarea utilizando la computadora.

1.1.3.1. Programa

Los programas son una serie de instrucciones escritas en un lenguaje de programación.

Los programas son ejecutados por la computadora para que ésta actúe de manera determinada.

1.1.3.2. Información

La información es el conocimiento que se obtiene por medio de conceptos, objetos, eventos, ideas, procesos, etc. La información en sentido de computación son instrucciones y datos.

Las instrucciones son código de un programa que ha sido interpretado o se ha compilado en lenguaje de máquina. Los datos son toda la información puesta en un formato procesable por la computadora.

1.1.4. Firmware

El firmware constituye programas guardados permanentemente en la computadora. Estos programas no se pueden modificar. Los programas son almacenados en *ROM* (*Read Only Memory*) o en otra parte como los chips del *BIOS*.

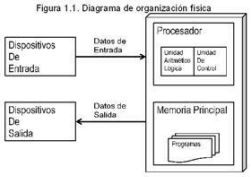
1.2. Organización de la Computadora

1.2.1. Diagrama de Organización Física

La computadora se encuentra integrada por cuatro componentes fundamentales. Estos componentes son los dispositivos de entrada, dispositivos de salida, el CPU y la memoria. La figura 1.1. muestra la organización física de una computadora.

Figura 11. Páginas 5 – 8 Libro “Programación principiantes”

Figura 1.1. Diagrama de organización física



1.2.2. Dispositivos E/S

Los dispositivos de entrada y salida son dispositivos que permiten la interacción entre el usuario y la computadora.

Los dispositivos de entrada son periféricos que permiten introducir información hacia la computadora.

Los dispositivos de salida son periféricos que permiten la representación de los resultados del procesamiento de los datos.

Los periféricos son dispositivos que están conectados a la computadora. La computadora es la encargada de controlar al dispositivo. Los periféricos son externos a la *CPU*.

1.2.3. *CPU*

CPU son las siglas para *central processing unit*. El *CPU* es considerado como el cerebro de la computadora. Esto es debido a que procesa la mayor parte de los cálculos.

El *CPU* se encuentra en un chip llamado microprocesador. El microprocesador es un circuito integrado. El *CPU* contiene la *ALU* (unidad aritmética lógica) y el *CU* (unidad de control). Los CPU algunas veces contienen la *FPU* (unidad de punto flotante).

ALU significa en inglés *unit arithmetic logic*. Esta es parte del *CPU*. La *ALU* toma las decisiones del microprocesador. Esto lo realiza por medio de la ejecución de operaciones aritméticas y lógicas.

CU significa en inglés *control unit*. Esta es parte del *CPU*. El *CU* obtiene instrucciones de programas. Este emite señales para realizar las instrucciones obtenidas.

La palabra es la unidad de información. Esta también se denomina en inglés *WORD*. Las palabras se componen de caracteres, bits o bytes. Estas son consideradas como una entidad. Las palabras se pueden almacenar en una localidad.

1.2.4. Memoria Intermedia (*caché*)

La memoria interna se denomina como *caché*. Esta memoria es la unidad pequeña de memoria ultra rápida. El tamaño de la memoria interna oscila entre 256 *kilobytes* a 512 *kilobytes*. La memoria interna almacena información recientemente accedida o que se accede con frecuencia. Esta memoria permite que el microprocesador no recupere esta información de circuitos de memoria más lentos.

1.2.5. Memoria Principal

La memoria principal son circuitos electrónicos capaces de almacenar información. El microprocesador tiene acceso a ellos. La velocidad de la memoria principal es extremadamente superior a la memoria secundaria. La capacidad de almacenamiento de la memoria principal es realmente inferior a la memoria secundaria.

La memoria principal posee dos tipos de memoria importantes. La ROM y la RAM.

ROM significa *read only memory*. Este tipo de memoria no pierde el contenido cuando se interrumpe la corriente eléctrica. El usuario ni el sistema pueden modificar la información de esta memoria.

RAM significa *random access memory*. Este tipo de memoria guarda instrucciones y datos de programas para que el *CPU* los acceda directamente

por el bus externo de alta velocidad. La *RAM* pierde su contenido si la corriente eléctrica se pierde.

1.2.5.1. Sistema Binario

La unidad elemental de memoria es el *byte*. Un *byte* es la unidad fundamental de información de las computadoras. El *byte* se encuentra compuesto de 8 bits. El *bit* es la unidad básica de información en un sistema binario. La representación de un *bit* es de 0 y 1.

El sistema binario es un sistema de numeración que se encuentra representado por los números 0 y 1. El sistema binario utiliza lógica binaria. Esto lo utiliza para poder formar sus operaciones y cantidades. El sistema binario es de base 2.

La tabla 1.1. contiene la medida y la representación de medidas de almacenamiento.

Tabla 1.1. Medidas de almacenamiento

Nombre	Abreviatura	Valor
Binary Digit	*BIT*	Unidad mínima de almacenamiento.
BYTE	*BYTE*	8 *BIT*
KILOBYTE	*KB*	1024 *BIT*
MEGABYTE	*MB*	1048576 *BIT*
GIGABYTE	*GB*	1073741824 *BIT*

1.2.6. Memoria Secundaria

La memoria secundaria son dispositivos periféricos orientados a almacenamiento masivo de datos. La capacidad de la memoria secundaria es muchísimo mayor a la de la memoria principal. La memoria secundaria referente a velocidad es muchísimo más lenta que la memoria principal.

Los tipos de dispositivos de memoria secundaria se catalogan en magnéticos, ópticos y memoria de destello. Los dispositivos magnéticos son las cintas o discos. El *DVD* (*digital versatile disk*) es un ejemplo de dispositivos ópticos.

1.2.7. Comparación entre Memorias

La memoria principal y la memoria secundaria difieren con respecto a la velocidad y a su capacidad de almacenamiento. Estas memorias se diferencian también en durabilidad, coste y tiempo de acceso. La tabla 1.2. muestra esta comparativa entre memorias.

Tabla 1.2. Comparativa de memorias

Tipo de Memoria / Características	Principal	Secundaria
Durabilidad	Volatil	Persistente
Capacidad	Limitada y relativamente baja (cada vez mayor)	Alta (cada vez mayor)
Velocidad	Rápido	Relativamente lento
Costo	Elevado	Bajo
Acceso directo desde el CPU	Si	No
Tiempo de acceso	Independiente de la posición.	Dependiente de la posición.

1.3. El Software

La sección 1.1.3. contiene lo concerniente al concepto de software. Esta sección trata sobre la división que posee el software.

1.3.1. División del Software

El software se divide en dos conceptos. El primero se denomina software del sistema. El segundo se denomina programas de utilidad.

El software del sistema son los programas necesarios para que la computadora funcione. El software del sistema se divide en tres tipos de programas. Estos son el sistema operativo, el editor de texto y el compilador/intérprete.

El sistema operativo es considerado como el programa de control. Este permite la administración de las funciones internas de la computadora. Esto debido a que dirige las operaciones globales de la computadora. El sistema

Figura 12. Páginas 9 – 12 Libro “Programación principiantes”

operativo brinda los medios de control para las operaciones. Esto es debido a que el sistema operativo le indica a la computadora para que ésta ejecute otros programas. El sistema operativo administra el sistema de archivos de la misma. Esto es porque el sistema operativo controla el almacenamiento y recuperación de archivos.

El compilador es un programa que permite leer instrucciones escritas en un lenguaje de programación entendible para humanos. El compilador permite posteriormente traducir esto a un programa ejecutable entendible para la máquina. El intérprete es un traductor de un lenguaje de programación de alto nivel. Este traduce y ejecuta el programa paralelamente. Un programa interpretado corre más lento que un programa compilado.

Los programas de utilidad permiten la facilidad de uso de la computadora. Estos programas permiten al usuario realizar cualquier tarea con la computadora de una forma sencilla. Los programas de utilidad pueden ser el editor de texto o procesador de palabras, editor de imágenes y cualquier programa que permite al usuario realizar tareas propias.

1.3.2. Relación entre Software del Sistema y Software de Aplicación

El software del sistema y el software de aplicación mantienen una relación de dependencia. Esto significa que el software de aplicación depende del software del sistema para poder funcionar.

1.4. Lenguajes de Programación

Los lenguajes de programación permiten crear programas. Esto es debido a que poseen formas en que se pueden escribir y leer instrucciones por personas.

Las instrucciones pueden en conjunto convertirse en programas y pueden ser entendidas por la computadora.

Los lenguajes de programación básicamente se dividen en lenguaje de máquina, lenguajes de bajo nivel y lenguajes de alto nivel.

1.4.1. Lenguajes de Máquina

El lenguaje de máquina se caracteriza por ser entendible por la máquina. Las instrucciones del lenguaje de máquina son cadenas binarias. El código de máquina es el código binario.

El código de máquina es de codificación difícil y lenta. La fiabilidad que posee este lenguaje es escasa. El mayor problema de programar en lenguaje de máquina es su gran dificultad de encontrar los errores. Los programas que se realicen en lenguaje de máquina son ejecutables solamente en el mismo tipo de *CPU* (en la misma máquina).

Los tipos de *CPU* en los que se puede realizar lenguaje de máquina son dos. Estos *CPU* contienen un conjunto de instrucciones que permiten el lenguaje de máquina. Los tipos de *CPU* son *CISC* y *RISC*.

CISC son las siglas de *complex instruction set computer*. Este tipo de *CPU* es capaz de reconocer 100 o más instrucciones. Estas son necesarias para realizar la mayor parte de los cálculos.

RISC son las siglas de *reduced instruction set computer*. Las instrucciones de este tipo de *CPU* son reducidas a un mínimo para aumentar la velocidad de procesamiento. Un *CPU RISC* se ejecuta más rápido que un *CPU CISC* debido a que posee menos instrucciones que interpretar.

1.4.2. Lenguajes de Bajo Nivel

Los lenguajes de bajo nivel describen de manera exacta los procedimientos que se seguirán en el *CPU*. Estos lenguajes son más fáciles de usar que los lenguajes de máquina. Los lenguajes de bajo nivel son dependientes del *CPU* como los lenguajes de máquina.

El lenguaje de bajo nivel por excelencia es el lenguaje ensamblador. Este lenguaje es cono conocido como *assembly language*. Las instrucciones en el lenguaje ensamblador corresponden cada una a una instrucción que el microprocesador puede llevar a cabo. El lenguaje ensamblador es un lenguaje de procedimientos. El código es compacto y opera rápidamente, más eficiente que uno de alto nivel.

El lenguaje ensamblador utiliza *mnemonics*. Los *mnemonics* son instrucciones del lenguaje ensamblador. *Mnemonics* típicos son: *ADD*, *SUB*, *DIV*, *MOV*, etc. Un programa escrito en ensamblador requiere una fase de traducción a un lenguaje de máquina.

1.4.3. Lenguaje de Alto Nivel

Los lenguajes de alto nivel se caracterizan por poseer un nivel de abstracción mayor que el del lenguaje ensamblador. Estos lenguajes permiten escribir programas fácil y rápido. El programa escrito en lenguaje de alto nivel debe ser traducido a lenguaje de máquina por un intérprete o un compilador.

Los programas escritos en lenguaje de alto nivel son independientes de la máquina donde se realicen. Los programas son portables y transportables. Los lenguajes de alto nivel permiten que la capacitación de los programadores sea corta en comparación con los otros lenguajes. El lenguaje de alto nivel se basa en reglas sintácticas semejantes al lenguaje humano. El costo de desarrollo por medio de estos lenguajes es bastante reducido.

El tiempo de ejecución de un programa de alto nivel es muchísimo mayor en comparación con programas hechos en otros lenguajes. Estos programas tienden a aumentar el uso de memoria. Los programas de alto nivel tienden al desaprovechamiento de los recursos internos de la máquina. Estos programas además deben pasar por diversas fases de traducción del programa fuente.

1.5. Algoritmo

Mohammed al-Khowarizmi junto con Euclides son considerados los padres de la algoritmia. Mohammed vivió en el siglo IX. Él fue un matemático persa. Euclides vivió en el siglo IV a.C. Él fue un matemático griego.

Un algoritmo es un procedimiento (método) para resolver problemas. Este procedimiento puede ser matemático o lógico.

El método permite dividir el problema en una serie de pasos sencillos.

Los algoritmos poseen tres características. Estas características permiten indicar que un algoritmo puede ser:

- Preciso: esto es debido a que se indica el orden en que se debe realizar cada paso del algoritmo.
- Definido: esto es debido a que si el algoritmo se sigue dos veces, se obtendrá el mismo resultado siempre.
- Finito: esto es debido a que el algoritmo debe poder terminar en algún momento.

Los pasos a macronivel para resolver un problema se pueden agrupar en tres. Estos son:

- Diseñar el algoritmo como una secuencia de pasos para solucionar el problema.
- Expresar el algoritmo como un programa en lenguaje de programación.
- Ejecutar y validar el programa por computadora.

1.5.1. Diferencia entre Métodos Algorítmicos y Heurísticos

Los métodos algorítmicos son propios de las computadoras. Estos consideran todas las alternativas o posibilidades de un problema. Los métodos algorítmicos obtienen una solución exacta. Esto contrasta con los métodos heurísticos. Los métodos heurísticos son propios del pensamiento humano. Estos utilizan una estrategia que aparenta ser la más adecuada a la solución. La solución encontrada por los métodos heurísticos no siempre es la mejor ni la más exacta.

1.6. Resolución de Problemas Computacionales

La resolución de un problema computacional conlleva una serie de fases asociadas que permiten realizar la solución al problema. Las fases que comprende la resolución de problemas computacionales son:

- Análisis del problema.
- Diseño del algoritmo.
- Codificación.
- Compilación y ejecución.
- Verificación y depuración.
- Documentación.
- Mantenimiento.

Figura 13. Páginas 13 – 16 Libro “Programación principiantes”

1.6.1. Análisis del Problema

El análisis del problema se refiere al análisis del problema considerando la especificación de los requisitos brindados por el cliente. Esta fase permite definir claramente lo que debe hacer el programa y cuál es el resultado deseado.

Al definir un problema se aconseja que para que la definición sea correcta, ésta debe responder a las siguientes preguntas:

- ¿Qué entradas se requieren? (tipo y cantidad)
- ¿Cuál es la salida deseada? (tipo y cantidad)
- ¿Qué método produce la salida deseada?

1.6.2. Diseño del Algoritmo

El diseño del algoritmo determina cómo hace el programa lo que tiene y debe realizar. Esta fase permite diseñar la solución que permitirá realizar un algoritmo que resuelva el problema. Los métodos eficaces de diseño de algoritmos se basan en el método conocido como *Divide & Conquer*.

El diseño de algoritmo puede utilizar el método de programación *top down*.

1.6.2.1. Introducción al Modelo Descendente (*Top Down*)

El modelo descendente es conocido también como modelo modular. El proceso de diseño inicia con el enunciado del propósito del programa. El propósito es dividido en un conjunto de sub categorías. Estas describen los aspectos de las funciones anticipadas del problema. Las sub categorías corresponden a un módulo de programa específico que se pude codificar de forma independiente.

El método descendente se conoce como modular debido a que éste divide el programa en módulos. Este método de diseño es flexible y potente. Los

módulos realizan una única actividad o tarea y son codificadas independientemente de otros módulos.

Un módulo es un subprograma que posee un solo punto de entrada y un solo punto de salida. Los módulos se dividen en dos. Estos son el módulo principal que es el módulo de nivel más alto que posee el control de todo. El otro módulo se conoce como sub módulo. Los sub módulos son módulos de nivel más bajo y se consideran como sub programas. El módulo principal llama a los sub programas y estos a su vez a otros sub programas.

El modelo modular permite independencia en la planificación, la codificación, la comprobación y la depuración del programa. Este modelo permite la integración debido a que pueden ser realizados los módulos independientemente y luego pueden ser combinados.

1.6.2.2. Herramientas de diseño

Un algoritmo se pude realizar utilizando dos herramientas de diseño. Estas herramientas son:

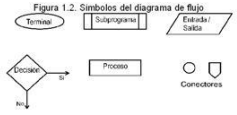
- Diagramas de flujo.
- Pseudocódigo.

Los diagramas de flujo son una representación gráfica de un algoritmo. Estos son utilizados para que se pueda apreciar gráficamente el comportamiento del algoritmo. Los diagramas de flujo utilizan una serie de símbolos. Los símbolos utilizados son:

- Terminal: éste se representa como una elipse.
- Subprograma: éste se representa como un rectángulo con dividido en tres utilizando líneas verticales.
- Entrada/Salida: éste se representa por medio de un romboide.
- Decisión: éste se representa por medio de un rombo.
- Proceso: éste se representa por medio de un rectángulo.
- Conectores: éste se representa por medio de un círculo o un trapezoide.

La figura 1.2. muestra la representación gráfica de cada uno de los símbolos utilizados en los diagramas de flujo.

Figura 1.2. Símbolos del diagrama de flujo



El pseudocódigo es la escritura de un programa en lenguaje que no corresponde a ningún lenguaje de programación. Esto debido a que su objetivo es simplemente facilitar la escritura y lectura del programa por personas que no entiendan un lenguaje de computación en particular. Esta independencia permite que la sintaxis sea lo más genérico posible.

1.6.3. Codificación

La codificación es transformar (escribir) el algoritmo obtenido en la fase de diseño a un lenguaje de programación.

La codificación implica seguir cada una de las reglas establecidas por el lenguaje de programación elegido.

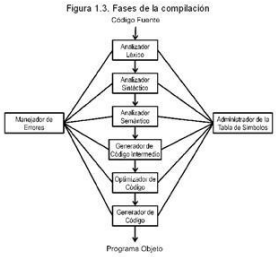
1.6.4. Compilación y Ejecución

La fase de compilación implica una serie de sub fases que se deben seguir desde que se obtiene el código fuente hasta que se obtiene el programa objeto.

El programa empieza con el código fuente. Este pasa posteriormente a una sub fase de análisis léxico. El análisis léxico permite verificar si el código fuente posee los símbolos reconocidos por el lenguaje de programación. La sub fase da paso a la sub fase de análisis sintáctico. Esto es para verificar si las operaciones y palabras corresponden a instrucciones referentes del lenguaje de programación. La sub fase siguiente es el análisis semántico. Este permite verificar si las instrucciones se encuentran lógicamente escritas para que el lenguaje de programación las entienda. La sub fase siguiente es el generador de código intermedio que permite generar el código. Esta permite paso a una fase que optimiza el código. La última sub fase corresponde al generador de código optimizado que permite obtener el programa objeto. Las sub fases

poseen un manejador de errores y un administrador de la tabla de símbolos. La figura 1.3. muestra las fases de la compilación.

Figura 1.3. Fases de la compilación



La fase compilación y de ejecución comprenden tres tipos de programas. Estos programas son el resultado de pasar por las sub fases. El programa fuente es el código que debe de ser traducido a lenguaje de máquina.

El programa objeto se obtiene luego de realizar la compilación del programa fuente. El programa ejecutable se produce por el proceso de *link* (carga del programa objeto con las librerías del compilador).

1.6.5. Verificación y Depuración

La fase de verificación es un proceso de ejecución de un programa con variedad de datos de prueba, que permitirán obtener *bugs* (errores) en caso de que existan.

La depuración es el proceso que permite encontrar los errores del programa y realizar su corrección o eliminación.

Figura 14. Páginas 17 – 20 Libro “Programación principiantes”

Los errores que se pueden encontrar son de tres tipos. Estos son:

- Errores de compilación: estos generalmente son errores de sintaxis. Estos ocurren por el uso incorrecto de las reglas de lenguaje de programación.
- Errores de ejecución: estos ocurren por instrucciones comprendidas pero que no es posible ejecutarlas.
- Errores lógicos: estos son errores de lógica del programa que conlleva al mal diseño del algoritmo.

1.6.6. Documentación

La documentación influye decisivamente el producto final. Esta permite describir los pasos que dieron lugar a la solución del problema. La documentación puede ser de dos tipos.

La documentación interna permite realizar comentarios dentro del programa fuente. Ésta permite indicar para qué sirve cada linea de código o cada segmento del código.

La documentación externa permite archivar el proceso de toda la solución. Este tipo de documentación incluye: el análisis, diagramas de flujo y/o pseudocódigo, diagrama de objetos, manuales de usuario, manuales técnicos, es decir cualquier documentación que permite el entendimiento del programa.

1.6.7. Mantenimiento

El mantenimiento se utiliza para corregir posibles errores o realizar un cambio en el programa o simplemente verificar que el programa funcione correctamente.

Una forma fácil de realizar el mantenimiento es mediante el versionamiento.

El versionamiento permite que al momento de realizar un programa, el código fuente sea identificado de tal manera que se lleve control sobre los cambios que se han realizado. Esto llevará a obtener distintos programas fuente desde una versión inicial hasta obtener la versión final.

Una versión es cada cambio realizado al código fuente.

1.7. Programación Estructurada

1.7.1. Definición, Reglas y Beneficios

La programación estructurada es un método de programación que permite obtener código legible, confiable y fácil de mantener. Esto lo busca realizar mediante tres estructuras:

- La estructura secuencial.
- La estructura selectiva.
- La estructura iterativa.

La programación estructurada se encuentra sujeta a ciertas reglas. Estas reglas son:

- El programa debe tener diseño modular.
- Los módulos deben estar diseñados descendentemente.
- Los módulos deben utilizar estructuras secuenciales, iterativas y selectivas.

Los beneficios que conlleva utilizar la programación estructurada es que se obtiene una reducción en el tiempo concerniente a: escribir, verificar, depurar y mantener los programas.

La programación estructurada utiliza un número limitado de estructuras de control que minimiza la complejidad de los programas y permite la reducción de los errores.

1.7.2. Diseño Descendente

El diseño descendente es un proceso en el que un problema es descompuesto en una serie de niveles. Esto significa que se realizan pasos sucesivos de refinamiento de la solución del problema.

El refinamiento sucesivo es dividir un problema en cada etapa y se expresa cada paso en una forma más detallada.

El problema se descompone en etapas o estructuras jerárquicas. Esto permite considerar dos perspectivas:

- ¿Qué hace?
- ¿Cómo lo hace?

1.7.3. Estructuras de Control

Las estructuras de control son métodos que permiten indicar el orden en el que las instrucciones del algoritmo se ejecutarán.

Las estructuras de control se dividen en 3 tipos básicos:

- Secuencia.
- Selección.
- Repetición.

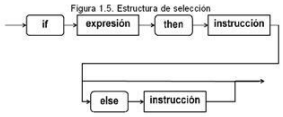
Las estructuras de secuencia permiten realizar una serie de instrucciones una después de otra en un orden descendente. La figura 1.4. representa el flujo de una secuencia de control de secuencia. Esta inicia, realiza la instrucción separada por punto y coma y sigue así hasta que ya no existan instrucciones y se procede a finalizar.

Figura 1.4. Estructura de secuencia

Las estructuras de control de selección permiten elegir entre una de dos opciones. Las opciones se pueden elegir solamente una de las dos pero no ambas.

La estructura de selección utiliza una expresión. Esta expresión es evaluada y de ser verdadera se realiza cierta acción o si es falsa se realiza la otra. La figura 1.5. muestra el flujo de una estructura de control de selección.

Figura 1.5. Estructura de selección



Las estructuras de repetición permiten ejecutar una secuencia varias veces. Esto se realiza hasta que una condición sea cumplida.

Existen dos tipos de estructuras de repetición. Estas son:

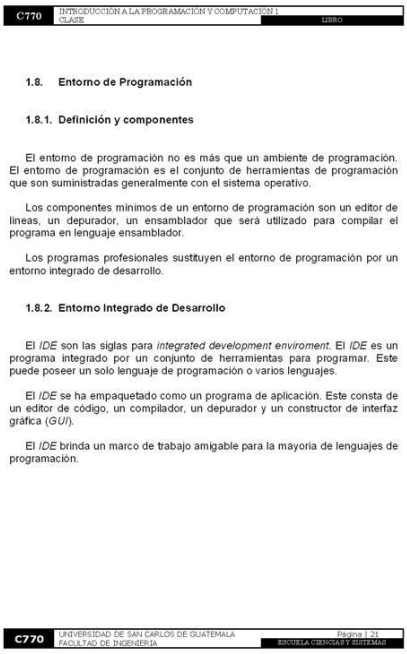
- La estructura for: esta estructura maneja una variable que toma valores en un rango especificado. Esto permite ejecutar las instrucciones un número de veces establecido.
- La estructura while: esta estructura maneja una expresión booleana que lleva el control del ciclo. Las instrucciones se repiten mientras la expresión sea verdadera.

1.7.4. Teorema de la Programación Estructurada

El teorema de la programación estructurada indica que un programa estructurado se puede escribir utilizando las tres estructuras de control básicas. Estas son estructuras secuenciales, de selección y de repetición.

El teorema indica que existe un solo punto de entrada y uno de salida para controlar el programa. El teorema también indica que hay caminos desde la entrada hasta la salida que recorren todas las partes del programa. El programa no debe contener ningún bucle o ciclo infinito.

Figura 15. Páginas 21 – 24 Libro “Programación principiantes”



1.8. Entorno de Programación

1.8.1. Definición y componentes

El entorno de programación no es más que un ambiente de programación. El entorno de programación es el conjunto de herramientas de programación que son suministradas generalmente con el sistema operativo.

Los componentes mínimos de un entorno de programación son un editor de líneas, un depurador, un ensamblador que será utilizado para compilar el programa en lenguaje ensamblador.

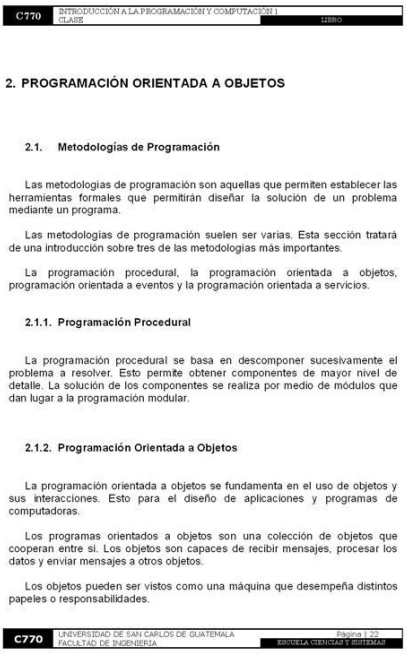
Los programas profesionales sustituyen el entorno de programación por un entorno integrado de desarrollo.

1.8.2. Entorno Integrado de Desarrollo

El *IDE* son las siglas para *integrated development enviroment*. El *IDE* es un programa integrado por un conjunto de herramientas para programar. Este puede poseer un solo lenguaje de programación o varios lenguajes.

El *IDE* se ha empaquetado como un programa de aplicación. Este consta de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (*GUI*).

El *IDE* brinda un marco de trabajo amigable para la mayoría de lenguajes de programación.



2. PROGRAMACIÓN ORIENTADA A OBJETOS

2.1. Metodologías de Programación

Las metodologías de programación son aquellas que permiten establecer las herramientas formales que permitirán diseñar la solución de un problema mediante un programa.

Las metodologías de programación suelen ser varias. Esta sección tratará de una introducción sobre tres de las metodologías más importantes.

La programación procedural, la programación orientada a objetos, programación orientada a eventos y la programación orientada a servicios.

2.1.1. Programación Procedural

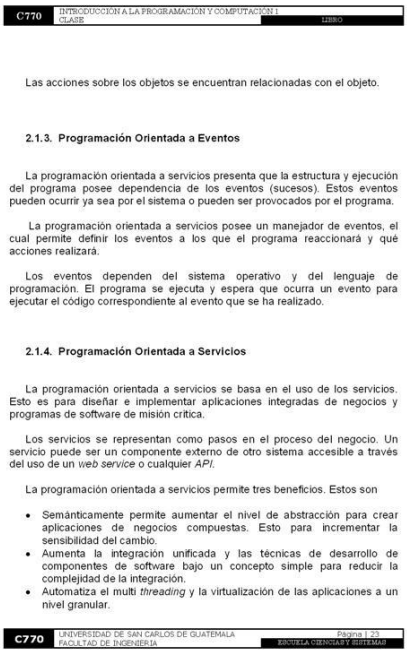
La programación procedural se basa en descomponer sucesivamente el problema a resolver. Esto permite obtener componentes de mayor nivel de detalle. La solución de los componentes se realiza por medio de módulos que dan lugar a la programación modular.

2.1.2. Programación Orientada a Objetos

La programación orientada a objetos se fundamenta en el uso de objetos y sus interacciones. Esto para el diseño de aplicaciones y programas de computadoras.

Los programas orientados a objetos son una colección de objetos que cooperan entre sí. Los objetos son capaces de recibir mensajes, procesar los datos y enviar mensajes a otros objetos.

Los objetos pueden ser vistos como una máquina que desempeña distintos papeles o responsabilidades.



Las acciones sobre los objetos se encuentran relacionadas con el objeto.

2.1.3. Programación Orientada a Eventos

La programación orientada a servicios presenta que la estructura y ejecución del programa posee dependencia de los eventos (sucesos). Estos eventos pueden ocurrir ya sea por el sistema o pueden ser provocados por el programa.

La programación orientada a servicios posee un manejador de eventos, el cual permite definir los eventos a los que el programa reaccionará y qué acciones realizará.

Los eventos dependen del sistema operativo y del lenguaje de programación. El programa se ejecuta y espera que ocurra un evento para ejecutar el código correspondiente al evento que se ha realizado.

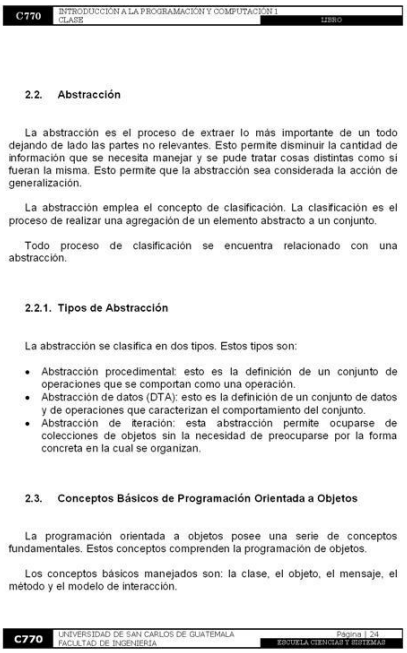
2.1.4. Programación Orientada a Servicios

La programación orientada a servicios se basa en el uso de los servicios. Esto es para diseñar e implementar aplicaciones integradas de negocios y programas de software de misión crítica.

Los servicios se representan como pasos en el proceso del negocio. Un servicio puede ser un componente externo de otro sistema accesible a través del uso de un *web service* o cualquier *API*.

La programación orientada a servicios permite tres beneficios. Estos son

- Semánticamente permite aumentar el nivel de abstracción para crear aplicaciones de negocios compuestas. Esto para incrementar la sensibilidad del cambio.
- Aumenta la integración unificada y las técnicas de desarrollo de componentes de software bajo un concepto simple para reducir la complejidad de la integración.
- Automatiza el multi *threading* y la virtualización de las aplicaciones a un nivel granular.



2.2. Abstracción

La abstracción es el proceso de extraer lo más importante de un todo dejando de lado las partes no relevantes. Esto permite disminuir la cantidad de información que se necesita manejar y se pude tratar cosas distintas como si fueran la misma. Esto permite que la abstracción sea considerada la acción de generalización.

La abstracción emplea el concepto de clasificación. La clasificación es el proceso de realizar una agregación de un elemento abstracto a un conjunto.

Todo proceso de clasificación se encuentra relacionado con una abstracción.

2.2.1. Tipos de Abstracción

La abstracción se clasifica en dos tipos. Estos tipos son:

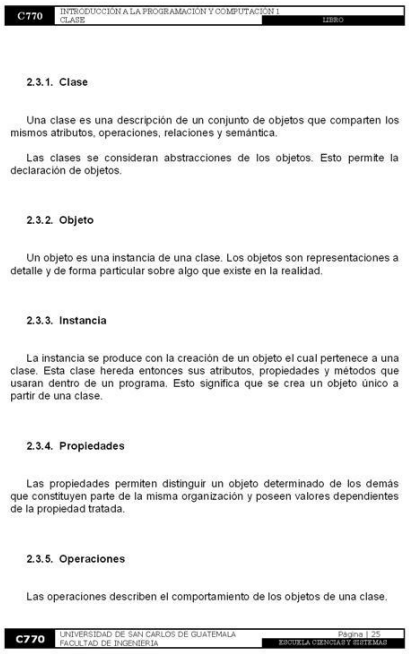
- Abstracción procedimental: esto es la definición de un conjunto de operaciones que se comportan como una operación.
- Abstracción de datos (DTA): esto es la definición de un conjunto de datos y de operaciones que caracterizan el comportamiento del conjunto.
- Abstracción de iteración: esta abstracción permite ocuparse de colecciones de objetos sin la necesidad de preocuparse por la forma concreta en la cual se organizan.

2.3. Conceptos Básicos de Programación Orientada a Objetos

La programación orientada a objetos posee una serie de conceptos fundamentales. Estos conceptos comprenden la programación de objetos.

Los conceptos básicos manejados son: la clase, el objeto, el mensaje, el método y el modelo de interacción.

Figura 16. Páginas 25 – 28 Libro “Programación principiantes”



2.3.1. Clase

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

Las clases se consideran abstracciones de los objetos. Esto permite la declaración de objetos.

2.3.2. Objeto

Un objeto es una instancia de una clase. Los objetos son representaciones a detalle y de forma particular sobre algo que existe en la realidad.

2.3.3. Instancia

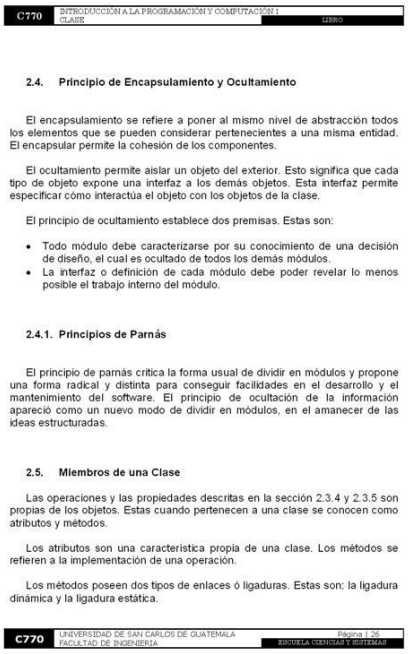
La instancia se produce con la creación de un objeto el cual pertenece a una clase. Esta clase hereda entonces sus atributos, propiedades y métodos que usaran dentro de un programa. Esto significa que se crea un objeto único a partir de una clase.

2.3.4. Propiedades

Las propiedades permiten distinguir un objeto determinado de los demás que constituyen parte de la misma organización y poseen valores dependientes de la propiedad tratada.

2.3.5. Operaciones

Las operaciones describen el comportamiento de los objetos de una clase.



2.4. Principio de Encapsulamiento y Ocultamiento

El encapsulamiento se refiere a poner al mismo nivel de abstracción todos los elementos que se pueden considerar pertenecientes a una misma entidad. El encapsular permite la cohesión de los componentes.

El ocultamiento permite aislar un objeto del exterior. Esto significa que cada tipo de objeto expone una interfaz a los demás objetos. Esta interfaz permite especificar cómo interactúa el objeto con los objetos de la clase.

El principio de ocultamiento establece dos premisas. Estas son:

- Todo módulo debe caracterizarse por su conocimiento de una decisión de diseño, el cual es ocultado de todos los demás módulos.
- La interfaz o definición de cada módulo debe poder revelar lo menos posible el trabajo interno del módulo.

2.4.1. Principios de Parnás

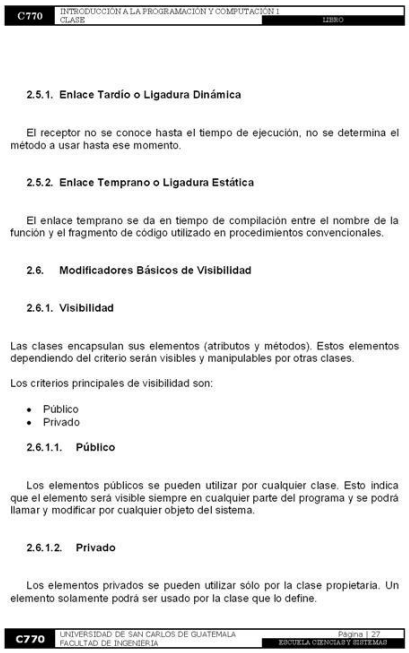
El principio de parnás critica la forma usual de dividir en módulos y propone una forma radical y distinta para conseguir facilidades en el desarrollo y el mantenimiento del software. El principio de ocultación de la información apareció como un nuevo modo de dividir en módulos, en el amanecer de las ideas estructuradas.

2.5. Miembros de una Clase

Las operaciones y las propiedades descritas en la sección 2.3.4 y 2.3.5 son propias de los objetos. Estas cuando pertenecen a una clase se conocen como atributos y métodos.

Los atributos son una característica propia de una clase. Los métodos se refieren a la implementación de una operación.

Los métodos poseen dos tipos de enlaces ó ligaduras. Estas son: la ligadura dinámica y la ligadura estática.



2.5.1. Enlace Tardío o Ligadura Dinámica

El receptor no se conoce hasta el tiempo de ejecución, no se determina el método a usar hasta ese momento.

2.5.2. Enlace Temprano o Ligadura Estática

El enlace temprano se da en tiempo de compilación entre el nombre de la función y el fragmento de código utilizado en procedimientos convencionales.

2.6. Modificadores Básicos de Visibilidad

2.6.1. Visibilidad

Las clases encapsulan sus elementos (atributos y métodos). Estos elementos dependiendo del criterio serán visibles y manipulables por otras clases.

Los criterios principales de visibilidad son:

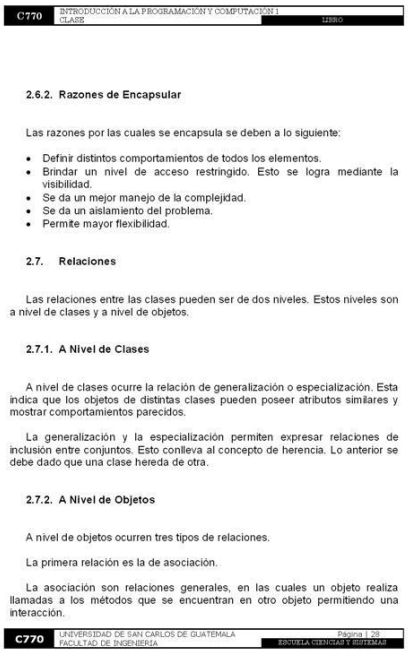
- Público
- Privado

2.6.1.1. Público

Los elementos públicos se pueden utilizar por cualquier clase. Esto indica que el elemento será visible siempre en cualquier parte del programa y se podrá llamar y modificar por cualquier objeto del sistema.

2.6.1.2. Privado

Los elementos privados se pueden utilizar sólo por la clase propietaria. Un elemento solamente podrá ser usado por la clase que lo define.



2.6.2. Razones de Encapsular

Las razones por las cuales se encapsula se deben a lo siguiente:

- Definir distintos comportamientos de todos los elementos.
- Brindar un nivel de acceso restringido. Esto se logra mediante la visibilidad.
- Se da un mejor manejo de la complejidad.
- Se da un aislamiento del problema.
- Permite mayor flexibilidad.

2.7. Relaciones

Las relaciones entre las clases pueden ser de dos niveles. Estos niveles son a nivel de clases y a nivel de objetos.

2.7.1. A Nivel de Clases

A nivel de clases ocurre la relación de generalización o especialización. Esta indica que los objetos de distintas clases pueden poseer atributos similares y mostrar comportamientos parecidos.

La generalización y la especialización permiten expresar relaciones de inclusión entre conjuntos. Esto conlleva al concepto de herencia. Lo anterior se debe dado que una clase hereda de otra.

2.7.2. A Nivel de Objetos

A nivel de objetos ocurren tres tipos de relaciones.

La primera relación es la de asociación.

La asociación son relaciones generales, en las cuales un objeto realiza llamadas a los métodos que se encuentran en otro objeto permitiendo una interacción.

Figura 17. Páginas 29 – 32 Libro “Programación principiantes”

La segunda relación es la agregación.

La agregación modela la relación parte/todo. Esto quiere decir una parte representa un elemento constituido por partes pequeñas.

La tercera relación es la composición.

La composición permite indicar que la existencia de las partes depende de la existencia del todo. Esto significa que si el todo es eliminado todas sus partes también son eliminadas.

2.8. Introducción a UML

2.8.1. Modelación Visual

Un modelo es una representación simple de la realidad. El modelar conlleva a dos tipos de modelado. El modelado del sistema y el modelado visual.

2.8.1.1. Modelado del Sistema

El modelado del sistema no es más que capturar todas las partes esenciales de un sistema.

2.8.1.2. Modelado Visual

El modelado visual es la simplificación del modelado del sistema abstrayéndolo y plasmándolo en una notación gráfica. *UML* se utiliza para el modelado de sistemas complejos, tanto en software como hardware.

El modelado visual es importante para capturar los procesos del negocio desde la perspectiva del usuario. El modelado visual analiza y diseña una aplicación destacando entre el dominio del negocio y el de la computadora. Este también reduce la complejidad y permite la independencia. El modelado visual promueve la reusabilidad de componentes.

2.8.2. *UML*

UML son las siglas para unified modeling language. *UML* es un lenguaje de modelado de sistemas de software que se encuentra respaldado por la *OMG* (*object management group*).

UML permite gráficamente visualizar, especificar, construir y documentar un sistema de software.

2.8.2.1. Evolución

UML surge por la necesidad de un lenguaje que permitiera transmitir y apoyar al proceso de análisis de un problema. Esto para poder transmitir la información a otros desarrolladores.

El año de 1994 se inicia el proceso de creación al unificar el método Booch y el OMT (object modeling tool).

La primera versión de UML es conocida como UML 0.8 y apareció en 1995. La versión que se utiliza actualmente es la UML 2.0 que apareció en el año 2002.

2.8.2.2. Diagramas

UML utiliza una serie de diagramas para representar el sistema del software. UML utiliza los siguientes diagramas:

- Diagramas de estructura: estos presentan la estructura estática del sistema. Entre estos se encuentran los diagramas de clases.
- Diagramas de comportamiento: estos presentan el comportamiento de los objetos del sistema. Entre estos se encuentran los diagramas de casos de uso, diagramas de estados y diagramas de actividades.
- Diagramas de interacción: estos presentan las interacciones que realizan los objetos en el sistema. Entre estos se encuentran los diagramas de secuencia y diagramas de colaboración.
- Diagramas de implementación: estos presentan el sistema desde el punto de vista físico. Entre estos se encuentran diagramas de componentes y diagramas de despliegue o ejecución.

2.8.2.3. Diagrama Estático de Clases

El diagrama estático de clases es un diagrama estático que describe la estructura del sistema denotando sus clases, atributos y las relaciones entre ellos.

El diagrama de clases utiliza los conceptos de clase, atributo, operación, método, objeto y las relaciones entre clases.

2.8.2.3.1. Clase

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Las clases se representan en el diagrama de clases como un rectángulo con tres divisiones internas.

Las clases se encuentran formadas por atributos. Los atributos son una característica propia de una clase. La sintaxis de un atributo es la siguiente:

- visibilidad nombre : tipo = [valor inicial], donde la visibilidad puede ser:
  - #: este indica que el atributo es protegido.
  - +: este indica que el atributo es público.
  - -: este indica que el atributo es privado.

Las clases también poseen operaciones y métodos. Las operaciones describen el comportamiento de los objetos de una clase. La sintaxis de una operación es la siguiente:

- visibilidad nombre (lista de parámetros)

Un método es una implementación de una operación. La sintaxis de un método es la siguiente:

- visibilidad nombre (lista de parámetros con tipo) : [valor entorno]

La notación correspondiente a una clase se presenta en la figura 2.1. La figura 2.1. presenta dos tipos de notación para una clase. La primera contiene

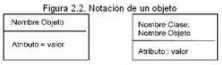
atributos sin definir y las operaciones. La segunda contiene atributos definidos y los métodos asociados.

Figura 2.1. Notación de una clase

2.8.2.3.2. Objeto

Un objeto es una instancia de una clase. La notación correspondiente a un objeto es que el nombre del objeto se coloque antes de los dos puntos. La notación corresponde a la de una clase. La diferencia existe en que el objeto posee valores, debido a que se refieren a un caso específico. La figura 2.2. presenta la notación de un objeto.

Figura 2.2. Notación de un objeto



Ejemplo 2.1.

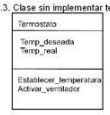
Un termostato es un componente que permite abrir o cerrar un circuito eléctrico en función de la temperatura para encender un ventilador. Se desea realizar una clase que presente una clase sin implementar, implementada y el objeto correspondiente.

Lo primero que se debe de realizar es establecer el nombre de la clase, en este caso será termostato. Lo siguiente es establecer los atributos. Los atributos identificados son temperatura deseada y temperatura real. Las operaciones que debe realizar la clase son establecer la temperatura y activar el ventilador.

La figura 2.3. muestra la clase sin implementar termostato. Esta clase posee bajo nivel de detalle.

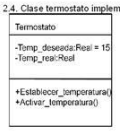
Figura 18. Páginas 33 – 36 Libro “Programación principiantes”

Figura 2.3. Clase sin implementar termostato



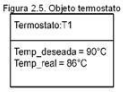
La figura 2.4. presenta la clase implementada termostato. Esta clase contiene la visibilidad propia de los atributos y los métodos.

Figura 2.4. Clase termostato implementada



La figura 2.5. muestra la clase termostato instanciada. Este objeto se llama T1 y sus atributos poseen valores que lo establecen como objeto.

Figura 2.5. Objeto termostato



2.8.2.3.3. Relaciones entre Clases

Las clases se relacionan entre si de distinta forma. Los tipos de relaciones que ocurren entre clases son:

- Asociación binaria.
- Asociación n-aria.
- Composición.
- Generalización.
- Dependencia.
- Relación de refinamiento.

Las relaciones se identifican como una línea. Las relaciones muchas veces poseen un rol.

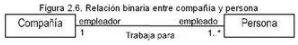
El rol se identifica como un nombre a los finales de la línea. El rol describe la semántica de la relación en el sentido indicado.

Las relaciones también incluyen la multiplicidad. La multiplicidad indica la cardinalidad de la relación. Los tipos de multiplicidad son:

- 1: esto significa uno.
- *: esto significa muchos.
- 1..*: esto significa uno o más.
- 0..*: esto significa cero o más.

La asociación binaria es una línea sólida que une dos clases. Este tipo de asociación no exige dependencia existencial, ni encapsulamiento. La figura 2.6. muestra la relación binaria entre una compañía y sus empleados. Esta relación posee do roles que corresponden a empleador y empleado. La multiplicidad del lado de compañía es de uno y del lado de persona es de uno o más.

Figura 2.6. Relación binaria entre compañía y persona



La figura 2.7. muestra la relación binaria entre una cuenta y un cliente. Esta relación binaria permite que un cliente sea el titular de una o más cuentas. La relación que existe es que un cliente es el titular de una cuenta.

Figura 2.7. Relación binaria entre cuenta y cliente

La asociación n-aria expresa una relación en tres o más clases. Esta relación se representa con un diamante del que salen líneas de asociación a las

clases. La figura 2.8. muestra una relación de los registros realizados por un jugador, en un equipo en una temporada.

Figura 2.8. Relación n-aria entre equipo, jugador y año

La figura 2.9. muestra la relación N-aria entre un avión, un piloto y un pasajero. Esto permite que un pasajero viajó en un avión, un piloto pilotea un avión y llevó pasajeros.

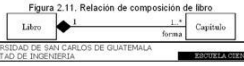
Figura 2.9. Relación N-aria entre avión, pasajero y piloto

La relación de composición exige una dependencia existencial. Esto es debido a que dentro de esta relación hay una pertenencia fuerte. Los objetos contenidos no son compartidos. Esta relación se denota dibujando un rombo relleno del lado de la clase que contiene la otra. La figura 2.10. presenta una relación de composición perteneciente a una ventana. Esta ventana es de una aplicación y se encuentra compuesta por un *header*, un panel y un *slider*.

Figura 210. Relación de composición de una ventana

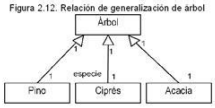
La figura 2.11. muestra la relación de composición que existe entre un libro y sus capítulos. Un libro está formado por uno o más capítulos. Uno o varios capítulos constituyen un libro.

Figura 2.11. Relación de composición de libro



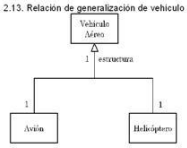
La relación de generalización denota la herencia entre las clases. Esta relación se representa por un triángulo sin rellenar en el lado de la superclase. Una superclase hereda a las subclases. La figura 2.12. presenta la relación de generalización de la clase árbol.

Figura 2.12. Relación de generalización de árbol



La figura 2.13. presenta la relación de generalización de la clase vehículo aéreo.

Figura 2.13. Relación de generalización de vehículo aéreo



La relación de dependencia es una relación semántica entre dos elementos del modelo. Esta relación indica cambiar el elemento independiente que pude requerir cambios en los dependientes. La relación de dependencia se representa con una línea punteada direccional, indicando el sentido de la dependencia.

2.9. Polimorfismo

La palabra polimorfismo deriva del griego. El significado de polimorfismo es poseer varias formas diferentes.

Figura 19. Páginas 37 – 40 Libro “Programación principiantes”

El polimorfismo es una característica de un lenguaje de programación, el cual logra permitir a los valores de diferentes tipos de datos ser manipulados usando una interfaz uniforme.

El polimorfismo permite tratar miembros de clases derivadas exactamente igual a sus padres miembros de clases. El polimorfismo es la habilidad de los objetos de pertenecer a diferentes tipos, para responder a llamadas de métodos del mismo nombre, cada uno acorde a un comportamiento apropiado de tipo específico.

El polimorfismo puede ocurrir de dos formas. El primer polimorfismo se denomina ad hoc. El segundo polimorfismo se denomina puro.

2.9.1. Polimorfismo *Ad Hoc*

El polimorfismo *ad hoc* también se conoce como polimorfismo de sobrecarga. Este polimorfismo se da al momento en que funciones con el mismo nombre, con funcionalidad similar en clases totalmente distintas. Esto quiere decir independientes una función de la otra.

El polimorfismo de sobre carga permite la definición de operadores en la cual el comportamiento varia dependiendo de los parámetros que se les aplican.

El polimorfismo de sobrecarga permite por ejemplo agregar al operador + funcionalidad distinta cuando hace referencia a una operación de dos números enteros o cuando se encuentra con dos cadenas de caracteres.

2.9.2. Polimorfismo Puro

El polimorfismo puro se conoce como polimorfismo paramétrico. Este polimorfismo es la capacidad en la cual se definen varias funciones empleando el mismo nombre, pero se utilizan parámetros distintos (nombre y/o tipo).

El polimorfismo puro selecciona automáticamente el método correcto a partir de los datos que se pasan en los parámetros.

Un ejemplo de este tipo de polimorfismo es el siguiente: El definir un método sumar que devuelva la suma de dos parámetros. Esto quiere decir que se definirá el método de esta forma:

- *float* sumar(*float, float*): devuelve la suma de dos número *float*.
- *int* sumar(*int, int*): devuelve la suma de dos números *int*.
- *char* sumar(*char, char*): devuelve el resultado de la suma de dos caracteres.

Esto permite indicar que dependiendo de los parámetros se elegirá el método adecuado para poder realizar la operación.

El polimorfismo se aplica debido a que los métodos poseen el mismo nombre pero cada uno realiza funciones distintas dependiendo sus parámetros.

2.10. Conceptos Avanzados

2.10.1. Prototipo de la Definición de Métodos

El definir métodos determina la funcionalidad de un objeto.

Los métodos constan particularmente de:

- Uno o varios modificadores.
- El objeto o tipo primitivo que se devuelve.
- El nombre del método.
- El conjunto de parámetros.
- La palabra reservada *throws*.
- El cuerpo del método.

La definición de métodos consta de dos conceptos importantes. El primero es la firma y el segundo es la sobrecarga.

La firma es conocida como *signature*. La firma es el tipo más pequeño de un método que sirve para identificar de manera única a un método. Esta debe ser única para cada método sobrecargado.

La firma incluye el número, tipo y el tipo de retorno.

La sobrecarga de métodos parte de crear varios métodos, los cuales poseen el mismo nombre pero ellos tienen firmas (*signature*) o definiciones diferentes.

2.10.2. Visibilidad y Modificadores de Acceso

Los modificadores de acceso definen el tipo de visibilidad que posee cada elemento de un objeto. Los modificadores de acceso pueden ser:

- Público: este permite q se pueda acceder al miembro de la clase desde cualquier lugar.
- Privado: este permite que se pueda acceder al miembro de la clase solamente desde la propia clase.
- Estático: este permite que se pueda acceder al miembro de la clase desde cualquier lugar pero no se puede modificar.
- Protegido: este permite que se pueda acceder al miembro de la clase solamente desde la propia clase o de una clase que herede de ella.

2.10.3. Miembros de Instancia

Los miembros de instancia comprenden las variables de ejemplares. Estas variables se conocen también como *instance variables*.

Una variable de ejemplar es una variable definida en una clase, para la cual cada objeto en la clase tiene una copia separada. Esto quiere decir que cada objeto tendrá distintos valores para estas variables.

2.10.4. Miembros Estáticos

Los miembros estáticos comprenden las variables de clase. Estas en inglés se llaman *class variables*. Las variables de clase se crean en la clase y cada objeto perteneciente a la clase la comparte. Esto quiere decir que esta variable es común para todos los objetos. Las variables de clase poseen el modificador *static* en su definición.

2.10.5. Definición de una Clase

Una clase para poder ser utilizada debe ser definida. Una clase se define (utilizando pseudocódigo) conforme se presenta en la figura 2.10.

Figura 2.10. Sintaxis de una clase en pseudocódigo

```
ModificadorAcceso ModificadorClase Clase
NombreClase IdentificadorHerencia
{
Atributo 1;
AtributoN;

Método 1;
Método N;
}
```

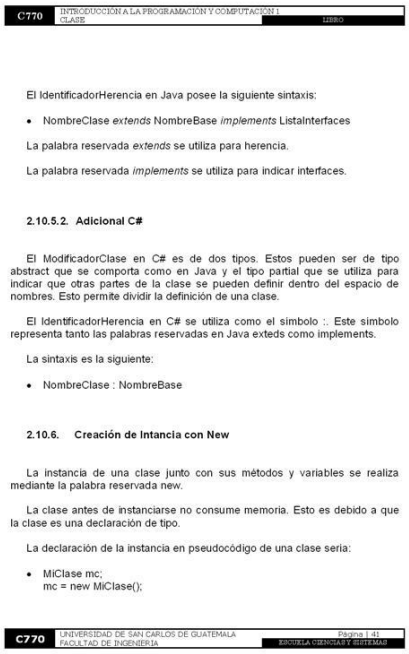
La definición en pseudocódigo comprende palabras reservadas. Estas palabras se describen a continuación.

- ModificadorAcceso: este permite especificar la visibilidad. La visibilidad puede ser pública, privada, protegida ó estática. La sección 2.10.2. comprende su detalle. El modificador de acceso si no se especifica la clase posee visibilidad de paquete y sólo puede ser utilizada por clases que se encuentren en el mismo paquete de esta clase.
- ModificadorClase: este comprende características propias de la clase. Los modificadores de clase pueden ser del tipo abstracto, final ó parcial.
- Clase: esta es la palabra reservada que indica que es una clase.
- NombreClase: este es el nombre que identificara a la clase.
- IdentificadorHerencia: el identificador de herencia variará dependiendo de las disposiciones a la hora de escribir pseudocódigo.
- Atributo N: este indica n atributos asociados a la clase.
- Método N: este indica n métodos asociados a la clase.

2.10.5.1. Adicional Java

El ModificadorClase en java puede ser de dos tipos. Estos son del tipo abstract que indica que falta al menos un código de algún método y del tipo final utilizado para evitar que la clase sea derivada.

Figura 20. Páginas 41 – 44 Libro “Programación principiantes”



El IdentificadorHerencia en Java posee la siguiente sintaxis:

- NombreClase *extends* NombreBase *implements* ListaInterfaces

La palabra reservada *extends* se utiliza para herencia.

La palabra reservada *implements* se utiliza para indicar interfaces.

2.10.5.2. Adicional C#

El ModificadorClase en C# es de dos tipos. Estos pueden ser de tipo abstract que se comporta como en Java y el tipo partial que se utiliza para indicar que otras partes de la clase se pueden definir dentro del espacio de nombres. Esto permite dividir la definición de una clase.

El IdentificadorHerencia en C# se utiliza como el simbolo :. Este simbolo representa tanto las palabras reservadas en Java exteds como implements.

La sintaxis es la siguiente:

- NombreClase : NombreBase

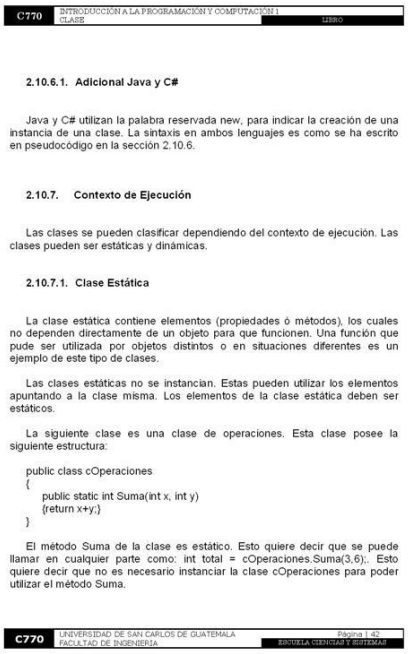
2.10.6. Creación de Intancia con New

La instancia de una clase junto con sus métodos y variables se realiza mediante la palabra reservada new.

La clase antes de instanciarse no consume memoria. Esto es debido a que la clase es una declaración de tipo.

La declaración de la instancia en pseudocódigo de una clase seria:

- MiClase mc;
  mc = new MiClase();



2.10.6.1. Adicional Java y C#

Java y C# utilizan la palabra reservada new, para indicar la creación de una instancia de una clase. La sintaxis en ambos lenguajes es como se ha escrito en pseudocódigo en la sección 2.10.6.

2.10.7. Contexto de Ejecución

Las clases se pueden clasificar dependiendo del contexto de ejecución. Las clases pueden ser estáticas y dinámicas.

2.10.7.1. Clase Estática

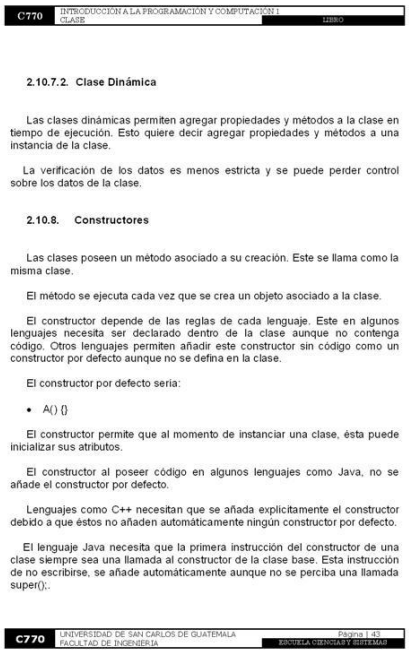
La clase estática contiene elementos (propiedades ó métodos), los cuales no dependen directamente de un objeto para que funcionen. Una función que pude ser utilizada por objetos distintos o en situaciones diferentes es un ejemplo de este tipo de clases.

Las clases estáticas no se instancian. Estas pueden utilizar los elementos apuntando a la clase misma. Los elementos de la clase estática deben ser estáticos.

La siguiente clase es una clase de operaciones. Esta clase posee la siguiente estructura:

```
public class cOperaciones
{
    public static int Suma(int x, int y)
    {return x+y;}
}
```

El método Suma de la clase es estático. Esto quiere decir que se puede llamar en cualquier parte como: int total = cOperaciones.Suma(3,6);. Esto quiere decir que no es necesario instanciar la clase cOperaciones para poder utilizar el método Suma.



2.10.7.2. Clase Dinámica

Las clases dinámicas permiten agregar propiedades y métodos a la clase en tiempo de ejecución. Esto quiere decir agregar propiedades y métodos a una instancia de la clase.

La verificación de los datos es menos estricta y se puede perder control sobre los datos de la clase.

2.10.8. Constructores

Las clases poseen un método asociado a su creación. Este se llama como la misma clase.

El método se ejecuta cada vez que se crea un objeto asociado a la clase.

El constructor depende de las reglas de cada lenguaje. Este en algunos lenguajes necesita ser declarado dentro de la clase aunque no contenga código. Otros lenguajes permiten añadir este constructor sin código como un constructor por defecto aunque no se defina en la clase.

El constructor por defecto seria:

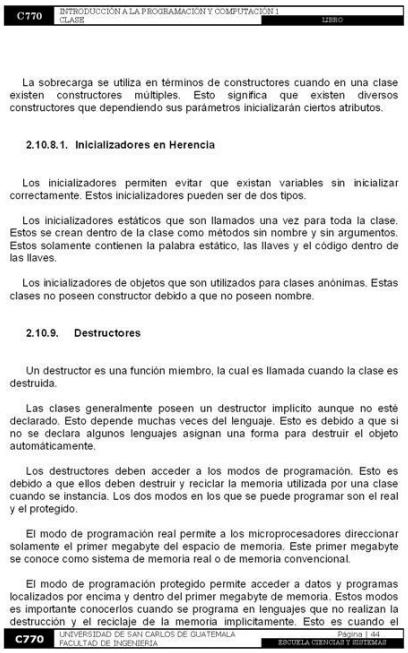
- A() {}

El constructor permite que al momento de instanciar una clase, ésta puede inicializar sus atributos.

El constructor al poseer código en algunos lenguajes como Java, no se añade el constructor por defecto.

Lenguajes como C++ necesitan que se añada explicitamente el constructor debido a que éstos no añaden automáticamente ningún constructor por defecto.

El lenguaje Java necesita que la primera instrucción del constructor de una clase siempre sea una llamada al constructor de la clase base. Esta instrucción de no escribirse, se añade automáticamente aunque no se perciba una llamada super();.



La sobrecarga se utiliza en términos de constructores cuando en una clase existen constructores múltiples. Esto significa que existen diversos constructores que dependiendo sus parámetros inicializarán ciertos atributos.

2.10.8.1. Inicializadores en Herencia

Los inicializadores permiten evitar que existan variables sin inicializar correctamente. Estos inicializadores pueden ser de dos tipos.

Los inicializadores estáticos que son llamados una vez para toda la clase. Estos se crean dentro de la clase como métodos sin nombre y sin argumentos. Estos solamente contienen la palabra estático, las llaves y el código dentro de las llaves.

Los inicializadores de objetos que son utilizados para clases anónimas. Estas clases no poseen constructor debido a que no poseen nombre.

2.10.9. Destructores

Un destructor es una función miembro, la cual es llamada cuando la clase es destruida.

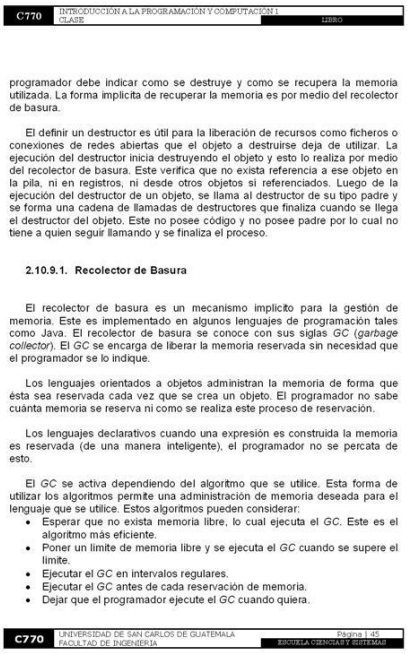
Las clases generalmente poseen un destructor implicito aunque no esté declarado. Esto depende muchas veces del lenguaje. Esto es debido a que si no se declara algunos lenguajes asignan una forma para destruir el objeto automáticamente.

Los destructores deben acceder a los modos de programación. Esto es debido a que ellos deben destruir y reciclar la memoria utilizada por una clase cuando se instancia. Los dos modos en los que se puede programar son el real y el protegido.

El modo de programación real permite a los microprocesadores direccionar solamente el primer megabyte del espacio de memoria. Este primer megabyte se conoce como sistema de memoria real o de memoria convencional.

El modo de programación protegido permite acceder a datos y programas localizados por encima y dentro del primer megabyte de memoria. Estos modos es importante conocerlos cuando se programa en lenguajes que no realizan la destrucción y el reciclaje de la memoria implicitamente. Esto es cuando el

Figura 21. Páginas 45 – 48 Libro “Programación principiantes”



programador debe indicar como se destruye y como se recupera la memoria utilizada. La forma implícita de recuperar la memoria es por medio del recolector de basura.

El definir un destructor es útil para la liberación de recursos como ficheros o conexiones de redes abiertas que el objeto a destruirse deja de utilizar. La ejecución del destructor inicia destruyendo el objeto y esto lo realiza por medio del recolector de basura. Este verifica que no exista referencia a ese objeto en la pila, ni en registros, ni desde otros objetos si referenciados. Luego de la ejecución del destructor de un objeto, se llama al destructor de su tipo padre y se forma una cadena de llamadas de destructores que finaliza cuando se llega el destructor del objeto. Este no posee código y no posee padre por lo cual no tiene a quien seguir llamando y se finaliza el proceso.

2.10.9.1. Recolector de Basura

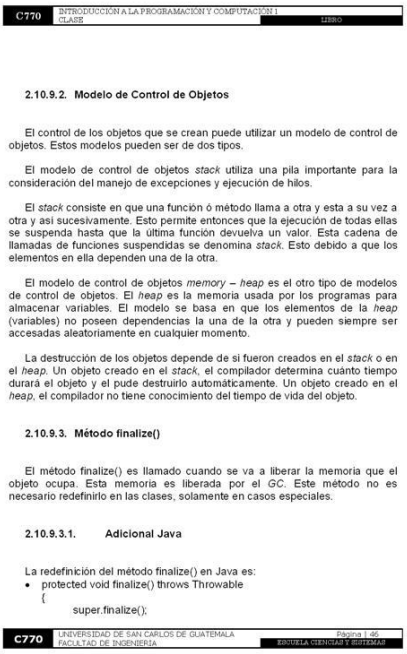
El recolector de basura es un mecanismo implícito para la gestión de memoria. Este es implementado en algunos lenguajes de programación tales como Java. El recolector de basura se conoce con sus siglas *GC* (*garbage collector*). El *GC* se encarga de liberar la memoria reservada sin necesidad que el programador se lo indique.

Los lenguajes orientados a objetos administran la memoria de forma que ésta sea reservada cada vez que se crea un objeto. El programador no sabe cuánta memoria se reserva ni como se realiza este proceso de reservación.

Los lenguajes declarativos cuando una expresión es construida la memoria es reservada (de una manera inteligente), el programador no se percata de esto.

El *GC* se activa dependiendo del algoritmo que se utilice. Esta forma de utilizar los algoritmos permite una administración de memoria deseada para el lenguaje que se utilice. Estos algoritmos pueden considerar:

- Esperar que no exista memoria libre, lo cual ejecuta el *GC*. Este es el algoritmo más eficiente.
- Poner un límite de memoria libre y se ejecuta el *GC* cuando se supere el límite.
- Ejecutar el *GC* en intervalos regulares.
- Ejecutar el *GC* antes de cada reservación de memoria.
- Dejar que el programador ejecute el *GC* cuando quiera.



2.10.9.2. Modelo de Control de Objetos

El control de los objetos que se crean puede utilizar un modelo de control de objetos. Estos modelos pueden ser de dos tipos.

El modelo de control de objetos *stack* utiliza una pila importante para la consideración del manejo de excepciones y ejecución de hilos.

El *stack* consiste en que una función ó método llama a otra y esta a su vez a otra y así sucesivamente. Esto permite entonces que la ejecución de todas ellas se suspenda hasta que la última función devuelva un valor. Esta cadena de llamadas de funciones suspendidas se denomina *stack*. Esto debido a que los elementos en ella dependen una de la otra.

El modelo de control de objetos *memory – heap* es el otro tipo de modelos de control de objetos. El *heap* es la memoria usada por los programas para almacenar variables. El modelo se basa en que los elementos de la *heap* (variables) no poseen dependencias la una de la otra y pueden siempre ser accesadas aleatoriamente en cualquier momento.

La destrucción de los objetos depende de si fueron creados en el *stack* o en el *heap*. Un objeto creado en el *stack*, el compilador determina cuánto tiempo durará el objeto y el pude destruirlo automáticamente. Un objeto creado en el *heap*, el compilador no tiene conocimiento del tiempo de vida del objeto.

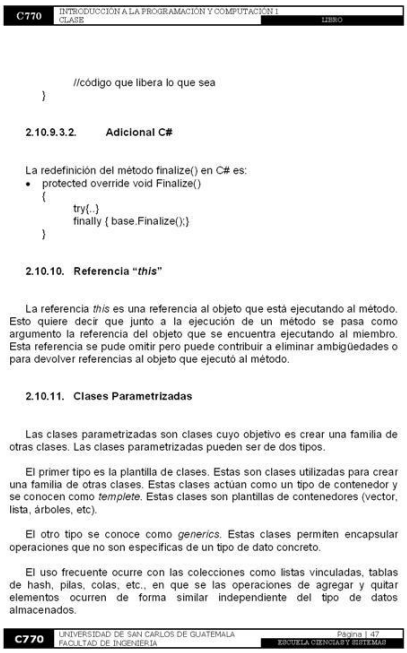
2.10.9.3. Método finalize()

El método finalize() es llamado cuando se va a liberar la memoria que el objeto ocupa. Esta memoria es liberada por el *GC*. Este método no es necesario redefinirlo en las clases, solamente en casos especiales.

2.10.9.3.1. Adicional Java

La redefinición del método finalize() en Java es:

- protected void finalize() throws Throwable
  {
  super.finalize();




```
//código que libera lo que sea
}
```

2.10.9.3.2. Adicional C#

La redefinición del método finalize() en C# es:

- protected override void Finalize()

```
{
    try{..}
    finally { base.Finalize();}
}
```

2.10.10. Referencia “*this*”

La referencia *this* es una referencia al objeto que está ejecutando al método. Esto quiere decir que junto a la ejecución de un método se pasa como argumento la referencia del objeto que se encuentra ejecutando al miembro. Esta referencia se pude omitir pero puede contribuir a eliminar ambigüedades o para devolver referencias al objeto que ejecutó al método.

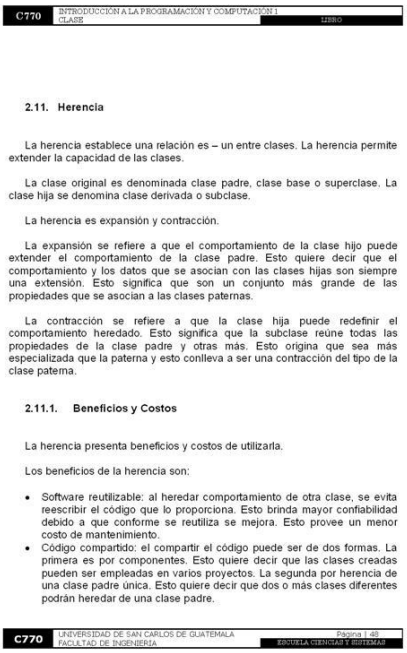
2.10.11. Clases Parametrizadas

Las clases parametrizadas son clases cuyo objetivo es crear una familia de otras clases. Las clases parametrizadas pueden ser de dos tipos.

El primer tipo es la plantilla de clases. Estas son clases utilizadas para crear una familia de otras clases. Estas clases actúan como un tipo de contenedor y se conocen como *templete*. Estas clases son plantillas de contenedores (vector, lista, árboles, etc).

El otro tipo se conoce como *generics*. Estas clases permiten encapsular operaciones que no son específicas de un tipo de dato concreto.

El uso frecuente ocurre con las colecciones como listas vinculadas, tablas de hash, pilas, colas, etc., en que se las operaciones de agregar y quitar elementos ocurren de forma similar independiente del tipo de datos almacenados.



2.11. Herencia

La herencia establece una relación es – un entre clases. La herencia permite extender la capacidad de las clases.

La clase original es denominada clase padre, clase base o superclase. La clase hija se denomina clase derivada o subclase.

La herencia es expansión y contracción.

La expansión se refiere a que el comportamiento de la clase hijo puede extender el comportamiento de la clase padre. Esto quiere decir que el comportamiento y los datos que se asocian con las clases hijas son siempre una extensión. Esto significa que son un conjunto más grande de las propiedades que se asocian a las clases paternas.

La contracción se refiere a que la clase hija puede redefinir el comportamiento heredado. Esto significa que la subclase reúne todas las propiedades de la clase padre y otras más. Esto origina que sea más especializada que la paterna y esto conlleva a ser una contracción del tipo de la clase paterna.

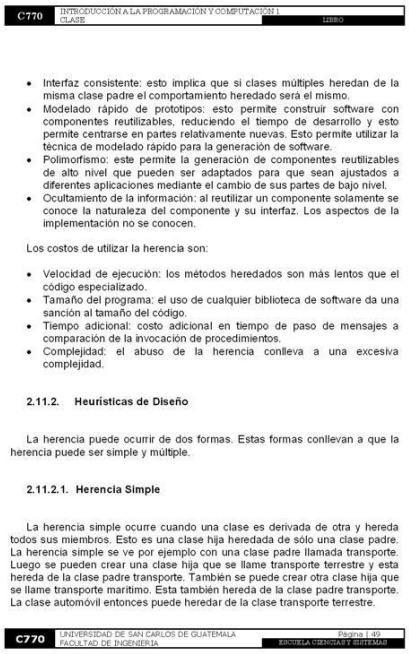
2.11.1. Beneficios y Costos

La herencia presenta beneficios y costos de utilizarla.

Los beneficios de la herencia son:

- Software reutilizable: al heredar comportamiento de otra clase, se evita reescribir el código que lo proporciona. Esto brinda mayor confiabilidad debido a que conforme se reutiliza se mejora. Esto provee un menor costo de mantenimiento.
- Código compartido: el compartir el código puede ser de dos formas. La primera es por componentes. Esto quiere decir que las clases creadas pueden ser empleadas en varios proyectos. La segunda por herencia de una clase padre única. Esto quiere decir que dos o más clases diferentes podrán heredar de una clase padre.

Figura 22. Páginas 49 – 52 Libro “Programación principiantes”



- Interfaz consistente: esto implica que si clases múltiples heredan de la misma clase padre el comportamiento heredado será el mismo.
- Modelado rápido de prototipos: esto permite construir software con componentes reutilizables, reduciendo el tiempo de desarrollo y esto permite centrarse en partes relativamente nuevas. Esto permite utilizar la técnica de modelado rápido para la generación de software.
- Polimorfismo: este permite la generación de componentes reutilizables de alto nivel que pueden ser adaptados para que sean ajustados a diferentes aplicaciones mediante el cambio de sus partes de bajo nivel.
- Ocultamiento de la información: al reutilizar un componente solamente se conoce la naturaleza del componente y su interfaz. Los aspectos de la implementación no se conocen.

Los costos de utilizar la herencia son:

- Velocidad de ejecución: los métodos heredados son más lentos que el código especializado.
- Tamaño del programa: el uso de cualquier biblioteca de software da una sanción al tamaño del código.
- Tiempo adicional: costo adicional en tiempo de paso de mensajes a comparación de la invocación de procedimientos.
- Complejidad: el abuso de la herencia conlleva a una excesiva complejidad.

2.11.2. Heurísticas de Diseño

La herencia puede ocurrir de dos formas. Estas formas conllevan a que la herencia puede ser simple y múltiple.

2.11.2.1. Herencia Simple

La herencia simple ocurre cuando una clase es derivada de otra y hereda todos sus miembros. Esto es una clase hija heredada de sólo una clase padre. La herencia simple se ve por ejemplo con una clase padre llamada transporte. Luego se pueden crear una clase hija que se llame transporte terrestre y esta hereda de la clase padre transporte. También se puede crear otra clase hija que se llame transporte marítimo. Esta también hereda de la clase padre transporte. La clase automóvil entonces puede heredar de la clase transporte terrestre.

2.11.2.2. Herencia Múltiple

Herencia múltiple se le llama también agregación o composición. Esta herencia consiste en crear una nueva clase que herede miembros de varias clases padre. Esto quiere decir que se hereda miembros de una o más clases base. La herencia múltiple se observa de la siguiente forma. Las clases transporte marítimo y transporte terrestre heredan de la clase padre transporte. Esto permite que la herencia múltiple sea para una clase overcraft. Este es un transporte que puede moverse tanto en el agua como en la tierra. Esto hace que la clase overcraft herede tanto de la clase transporte marítimo y su clase transporte terrestre.

2.11.3. Adicional

2.11.3.1. Java

La herencia en java se maneja por medio de la palabra reservada *extends*. Esta sirve para derivar una clase hija de una clase padre.

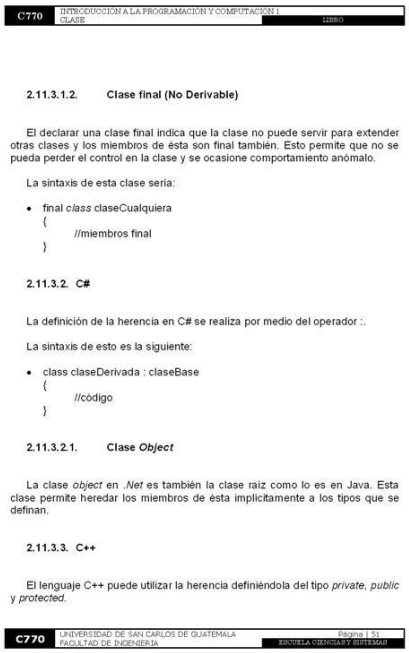
La sintaxis aplicada a Java sería:

- *class* claseDerivada *extends* claseBase
  {
  //código de la clase
  }

2.11.3.1.1. Clase *Object* (Raíz)

La clase *object* es la clase raíz de todo el árbol jerárquico de clases de Java. Esta clase brinda métodos útiles que pueden utilizar todos los objetos en java. Entre estos se encuentran:

- Método de comparación de un objeto a otro.
- Método de conversión de objeto a cadena.
- Método para esperar que ocurra una determinada condición.



2.11.3.1.2. Clase final (No Derivable)

El declarar una clase final indica que la clase no puede servir para extender otras clases y los miembros de ésta son final también. Esto permite que no se pueda perder el control en la clase y se ocasione comportamiento anómalo.

La sintaxis de esta clase sería:

- final *class* claseCualquiera
  {
  //miembros final
  }

2.11.3.2. C#

La definición de la herencia en C# se realiza por medio del operador :.

La sintaxis de esto es la siguiente:

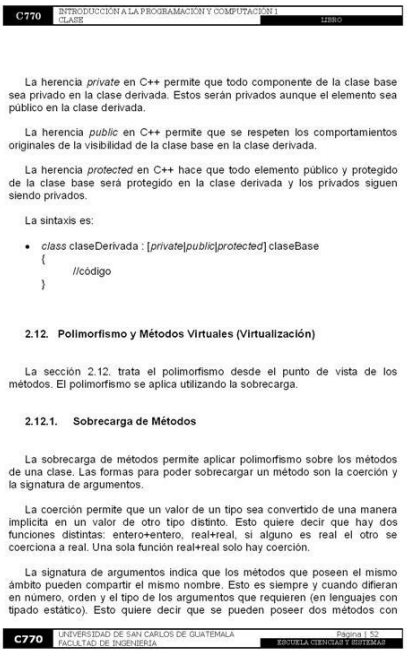
- class claseDerivada : claseBase
  {
  //código
  }

2.11.3.2.1. Clase *Object*

La clase *object* en *.Net* es también la clase raíz como lo es en Java. Esta clase permite heredar los miembros de ésta implícitamente a los tipos que se definan.

2.11.3.3. C++

El lenguaje C++ puede utilizar la herencia definiéndola del tipo *private, public* y *protected*.



La herencia *private* en C++ permite que todo componente de la clase base sea privado en la clase derivada. Estos serán privados aunque el elemento sea público en la clase derivada.

La herencia *public* en C++ permite que se respeten los comportamientos originales de la visibilidad de la clase base en la clase derivada.

La herencia *protected* en C++ hace que todo elemento público y protegido de la clase base será protegido en la clase derivada y los privados siguen siendo privados.

La sintaxis es:

- *class* claseDerivada : [*private*|*public*|*protected*] claseBase
  {
  //código
  }

2.12. Polimorfismo y Métodos Virtuales (Virtualización)

La sección 2.12. trata el polimorfismo desde el punto de vista de los métodos. El polimorfismo se aplica utilizando la sobrecarga.

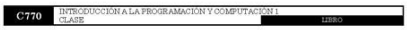
2.12.1. Sobrecarga de Métodos

La sobrecarga de métodos permite aplicar polimorfismo sobre los métodos de una clase. Las formas para poder sobrecargar un método son la coerción y la signatura de argumentos.

La coerción permite que un valor de un tipo sea convertido de una manera implícita en un valor de otro tipo distinto. Esto quiere decir que hay dos funciones distintas: entero+entero, real+real, si alguno es real el otro se coerciona a real. Una sola función real+real solo hay coerción.

La signatura de argumentos indica que los métodos que poseen el mismo ámbito pueden compartir el mismo nombre. Esto es siempre y cuando difieran en número, orden y el tipo de los argumentos que requieren (en lenguajes con tipado estático). Esto quiere decir que se pueden poseer dos métodos con

Figura 23. Páginas 53 – 56 Libro “Programación principiantes”



nombre igual a sumar. La diferencia entre ambos es que el primer método posee dos argumentos enteros. Este evalúa y devuelve la suma de estos valores. El segundo método posee dos cadenas de caracteres como argumentos. Este realiza la concatenación de la segunda cadena en la primera cadena. La signatura de argumentos permite entonces que dos métodos con nombre igual realicen funciones distintas dependiendo de la signatura de sus argumentos.

2.12.2. Sobrecarga de Operadores

La sobrecarga de operadores permite el uso de operadores tradicionales con tipos que son definidos por el usuario.

Un operador se sobrecarga con el símbolo @ es decir:

- <tipo devuelto> *operator*@ (<argumentos>)

Un operador con un objeto de tipo definido por el usuario para utilizarse se debe sobrecargar.

La sobrecarga de operadores no permite cambiar:

- Precedencia.
- Asociatividad (a=b=c -> a=(b=c)).
- Aridad (operador binario no puede actuar como unario o viceversa).

La sobrecarga de operadores no incluye los tipos predefinidos. Esto quiere decir que teniendo la expresión float resultado = int val1 + int val2; el resultado debe ser un tipo flotante, pero en la operación se están sumando dos valores enteros. La coerción permite que el valor obtenido por sumar los dos enteros sea devuelto a la variable resultado como un tipo flotante. Esto convierte el valor de tipo entero a flotante.

2.12.3. Anulación

La anulación permite a una subclase anular un método de la superclase por dos motivos. Estos motivos son el reemplazo y el refinamiento.

El refinamiento consiste en que el método de la clase hija posee el método del padre más acciones específicas. Los métodos de inicialización son un ejemplo de este tipo.

El reemplazo permite mejorar la implementación. Esto quiere decir que se puede redefinir un método en la clase hija. El redefinir el método área en la clase triángulo es un ejemplo de reemplazo.

2.12.4. Métodos Virtuales

Un método virtual es un método que es declarado virtual y por esto su comportamiento se encuentra determinado por la definición de un método con la misma cabecera en algunas de las subclases.

Los métodos virtuales pueden ser de dos tipos. Los métodos virtuales puros y los métodos virtuales diferidos.

Los métodos virtuales puros se deben implementar por una subclase que no sea abstracta. Las clases abstractas son las que contienen métodos virtuales puros. Los métodos virtuales puros generalmente poseen una declaración (cabecera) pero no contienen definición (implementación).

Los métodos virtuales diferidos se conocen también como genéricos. Estos métodos virtuales sirven para cualquier tipo sin necesidad de tener una función duplicada. Estos métodos permiten ser definidos sin la necesidad de especificar el tipo de todos o algunos de sus argumentos.

2.12.5. Adicional

2.12.5.1. Java

Java utiliza la palabra reservada final también en métodos. Esta palabra reservada se utiliza para evitar la sobrecarga.

Un método final es un método que no puede ser redefinido en una subclase de la clase en que fue definido. Las razones para realizar un método de este tipo son el rendimiento y el diseño. La sintaxis de este es:

- visibilidad final Tipo NombreMétodo (argumentos)

2.12.5.2. C#

C# también puede evitar la sobrecarga en los métodos. Esto hace que una subclase no pueda redefinir el método de la clase hija. C# necesita que se utilice la palabra reservada sealed para indicar que el método es del tipo final. La sintaxis es la siguiente:

- visibilidad sealed tipo NombreMétodo (argumentos)

2.13. Construcciones Abstractas

2.13.1. Clase Abstracta

Las clases abstractas se conocen también como diferidas. Una clase abstracta es aquella que no se puede instanciar. La clase abstracta es utilizada para definir subclases.

Las clases abstractas se utilizan cuando se desea que se defina una abstracción que contenga objetos de distintos tipos y por esto hacen uso del polimorfismo.

La clase es abstracta cuando uno de sus métodos no tiene implementación.

2.13.2. Métodos Abstractos

Un método abstracto es un método que ha sido declarado en una clase para el cual la clase no brinda la implementación para el método.

2.13.3. Interfaces

Una interface es una clase totalmente abstracta. Esto quiere decir que una interface es una clase sin implementación.

2.13.3.1. Diferencia entre Herencia e Implementación de Interface

La implementación de una interface implica dos cosas:

- Lo único que puede aparecer es:
  - Declaraciones de los métodos (nombre y signatura, sin su implementación).
  - Definición de constantes simbólicas.
- No encapsular los datos, debido a que sólo define los métodos que han de implementar los objetos de las clases que implementen la interfaz.

La clase que implementa una interfaz debe definir todos los métodos definidos en la interfaz.

La clase que hereda de una clase padre no necesariamente debe definir los métodos de su clase padre, solamente los que necesite sobrecargar.

2.13.4. Similitudes y Diferencias entre Clases Abstractas e Interfaces.

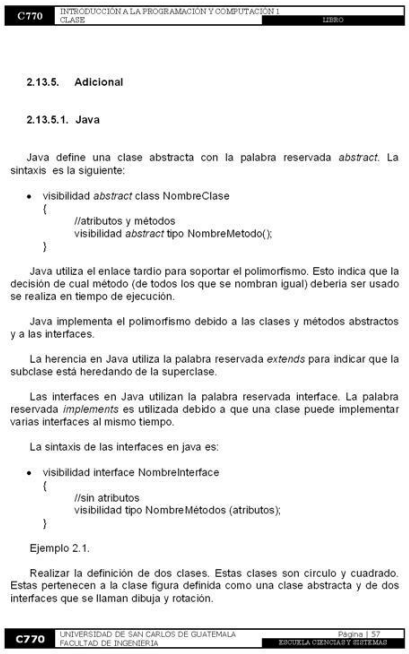
Las similitudes entre una clase abstracta y una interface son:

- Ninguna puede crear instancias. Esto quiere decir que no se pueden utilizar directamente para crear un objeto.
- No pueden ser selladas. Esto quiere decir que de sellar una interface ésta no se pude implementar.

Las diferencias entre las interfaces y las clases abstractas son:

- Las interfaces:
  - No contienen implementaciones.
  - No declaran miembros no públicos.
  - Utilizada para ampliar otras interfaces.
- Las clases abstractas:
  - Contienen implementaciones.
  - Declaran miembros no públicos.
  - Amplía otras clases que pueden ser no abstractas.

Figura 24. Páginas 57 – 60 Libro “Programación principiantes”



2.13.5. Adicional

2.13.5.1. Java

Java define una clase abstracta con la palabra reservada *abstract*. La sintaxis es la siguiente:

- visibilidad *abstract* class NombreClase
  {
  //atributos y métodos
  visibilidad *abstract* tipo NombreMetodo();
  }

Java utiliza el enlace tardío para soportar el polimorfismo. Esto indica que la decisión de cual método (de todos los que se nombran igual) debería ser usado se realiza en tiempo de ejecución.

Java implementa el polimorfismo debido a las clases y métodos abstractos y a las interfaces.

La herencia en Java utiliza la palabra reservada *extends* para indicar que la subclase está heredando de la superclase.

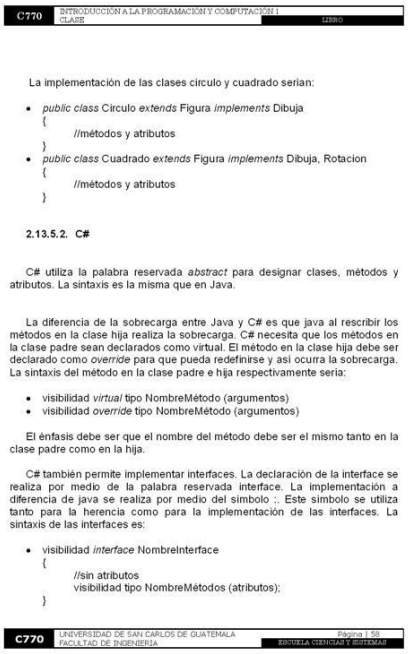
Las interfaces en Java utilizan la palabra reservada interface. La palabra reservada *implements* es utilizada debido a que una clase puede implementar varias interfaces al mismo tiempo.

La sintaxis de las interfaces en java es:

- visibilidad interface NombreInterface
  {
  //sin atributos
  visibilidad tipo NombreMétodos (atributos);
  }

Ejemplo 2.1.

Realizar la definición de dos clases. Estas clases son circulo y cuadrado. Estas pertenecen a la clase figura definida como una clase abstracta y de dos interfaces que se llaman dibuja y rotación.



La implementación de las clases circulo y cuadrado serían:

- *public class* Circulo *extends* Figura *implements* Dibuja
  {
  //métodos y atributos
  }
- *public class* Cuadrado *extends* Figura *implements* Dibuja, Rotacion
  {
  //métodos y atributos
  }

2.13.5.2. C#

C# utiliza la palabra reservada *abstract* para designar clases, métodos y atributos. La sintaxis es la misma que en Java.

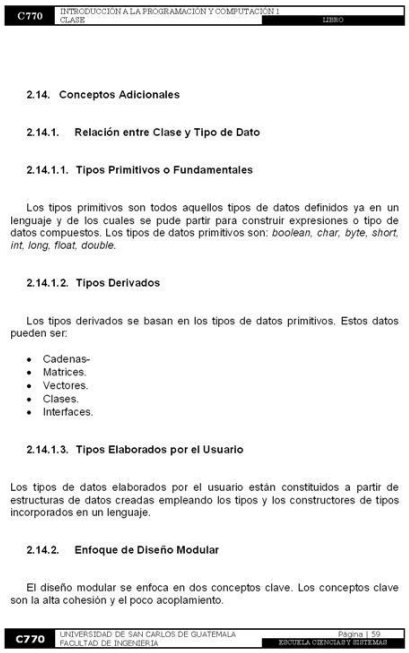
La diferencia de la sobrecarga entre Java y C# es que java al rescribir los métodos en la clase hija realiza la sobrecarga. C# necesita que los métodos en la clase padre sean declarados como virtual. El método en la clase hija debe ser declarado como *override* para que pueda redefinirse y así ocurra la sobrecarga. La sintaxis del método en la clase padre e hija respectivamente sería:

- visibilidad *virtual* tipo NombreMétodo (argumentos)
- visibilidad *override* tipo NombreMétodo (argumentos)

El énfasis debe ser que el nombre del método debe ser el mismo tanto en la clase padre como en la hija.

C# también permite implementar interfaces. La declaración de la interface se realiza por medio de la palabra reservada interface. La implementación a diferencia de java se realiza por medio del símbolo :. Este símbolo se utiliza tanto para la herencia como para la implementación de las interfaces. La sintaxis de las interfaces es:

- visibilidad *interface* NombreInterface
  {
  //sin atributos
  visibilidad tipo NombreMétodos (atributos);
  }



2.14. Conceptos Adicionales

2.14.1. Relación entre Clase y Tipo de Dato

2.14.1.1. Tipos Primitivos o Fundamentales

Los tipos primitivos son todos aquellos tipos de datos definidos ya en un lenguaje y de los cuales se pude partir para construir expresiones o tipo de datos compuestos. Los tipos de datos primitivos son: *boolean, char, byte, short, int, long, float, double.*

2.14.1.2. Tipos Derivados

Los tipos derivados se basan en los tipos de datos primitivos. Estos datos pueden ser:

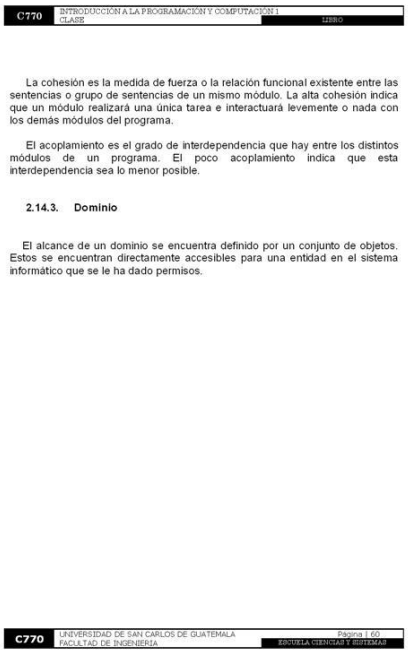
- Cadenas-
- Matrices.
- Vectores.
- Clases.
- Interfaces.

2.14.1.3. Tipos Elaborados por el Usuario

Los tipos de datos elaborados por el usuario están constituidos a partir de estructuras de datos creadas empleando los tipos y los constructores de tipos incorporados en un lenguaje.

2.14.2. Enfoque de Diseño Modular

El diseño modular se enfoca en dos conceptos clave. Los conceptos clave son la alta cohesión y el poco acoplamiento.



La cohesión es la medida de fuerza o la relación funcional existente entre las sentencias o grupo de sentencias de un mismo módulo. La alta cohesión indica que un módulo realizará una única tarea e interactuará levemente o nada con los demás módulos del programa.

El acoplamiento es el grado de interdependencia que hay entre los distintos módulos de un programa. El poco acoplamiento indica que esta interdependencia sea lo menor posible.

2.14.3. Dominio

El alcance de un dominio se encuentra definido por un conjunto de objetos. Estos se encuentran directamente accesibles para una entidad en el sistema informático que se le ha dado permisos.

Figura 25. Páginas 61 – 64 Libro “Programación principiantes”

3. INTRODUCCIÓN A LA PLATAFORMA JAVA

3.1. Evolución del Lenguaje Java

El lenguaje java ha evolucionado tomando partes de distintos lenguajes desde la primera aparición del lenguaje Java.

El lenguaje que inició todo fue el lenguaje B, pasando por el lenguaje C, posteriormente el lenguaje C++.

3.1.1. Lenguaje B

El lenguaje B fue desarrollado por Bell Labs. El máximo desarrollador fue Ken Thompson. Este lenguaje apareció en el año 1969.

El lenguaje B se encuentra influenciado por BCPL (*basic combined programming language*).

El lenguaje B es una simplificación de éste lenguaje. El lenguaje B tiene un tipo de datos que corresponde a una palabra de máquina.

3.1.2. Lenguaje C

El lenguaje C fue creado en 1972 por Ken Thompson y Dennis M. Ritchie en Bell Labs. El lenguaje C al igual que el lenguaje B se enfoca en la implementación de sistemas operativos.

El lenguaje C es débilmente tipificado de medio nivel con muchas características de bajo nivel.

El lenguaje C es eficiente en el sentido que permite utilizar las características de bajo nivel para óptimamente realizar implementaciones. Los compiladores de este lenguaje existen para casi todos los sistemas.

El lenguaje C permite realizar programas modulares y utilizar código o bibliotecas existentes.

3.1.2.1. UNIX

UNIX es un sistema operativo que fue desarrollado en sus inicios en lenguaje ensamblador.

El *kernel* de UNIX en el año de 1973 se encontraba escrito en lenguaje C la mayor parte. Este *kernel* fue uno de los primeros en implementarse en un lenguaje distinto al ensamblador.

3.1.2.2. K&R

Ritchie y Brian Kernighan publicaron la primera edición de El Lenguaje de Programación C en 1978. Ellos introducen al lenguaje las siguientes modificaciones:

- Tipo de datos *struct*.
- Tipo de datos *long int*.
- Tipo de datos *unsigned int*.
- Operadores =+ y =- sustituidos por += y -=.

El C de K&R es el más básico subconjunto del lenguaje que un compilador puede soportar.

3.1.2.3. ANSI C

ANSI C es la primera estandarización del lenguaje C. Esta estandarización se nombró como ANSI X3.159-1989. Esta fue realizada por el comité X3j11 nombrado por el Instituto Nacional Estadounidense de Estándares.

Al ANSI C se le realizan posteriormente modificaciones por parte de la ISO y se crea el estándar ISO/IEC 9899:1990, conocida como C90. Estas estandarizaciones buscan extender el C de K&R con funcionalidades no oficiales y por ende nuevas funcionalidades. Las nuevas funcionalidades fueron como prototipos de función y un preprocesador mejorado.

3.1.3. Lenguaje C++

El lenguaje C++ es un lenguaje híbrido. Esto debido a que permite la programación estructurada y programación orientada a objetos. Este lenguaje no posee verificación de errores en tiempo de ejecución.

El código generado por el compilador no debe incluir sobrecarga de recursos. Esto minimiza la implementación de recursos como polimorfismo y expansión de patrones.

El compilador de C++ genera código nativo con alta optimización en memoria y velocidad. El lenguaje C++ apoya el desarrollar clases genéricas con parámetros de tamaño y tipo por medio de los patrones de clases y funciones.

3.1.3.1. ANSI/ISO C++

La ISO/IEC 14882:1998 en el año de 1998, permitió estandarizar el lenguaje C++. El estándar permitió:

- Eliminar la extensión de los ficheros de *#include*.
- Utilizar *namespaces*.
- Considerar el uso de la librería stdio como anticuado.
- Añadir las clases istream y ostream.

3.1.4. Lenguaje Java

El nombre de Java fue acuñado por sus creadores James Gosling, Arthur Van Hoff, Andy Bechtosheim. El lenguaje Java surge posteriormente al proyecto *green*.

El equipo de creadores a mediados de 1994 reorienta Java hacia la web. Esto permite la creación de WebRunner que más tarde se convertiría en HotJava.

3.1.4.1. Proyecto *Green*

Sun Microsystem en el año de 1991 inicia una operación denominada proyecto *green*, en la cual se crea la tecnología Java para ser usada en un proyecto *set-top-box*.

El equipo del proyecto incluía trece personas y fue dirigido por James Gosling, durante 18 meses que duró el desarrollo.

3.1.4.2. *Oak*

Oak fue el nombre inicial del lenguaje producto del proyecto *green*. El nombre *oak* se debe a un roble que se encontraba afuera de la oficina de Gosling. El nombre posteriormente se cambia a *green* debido a que *oak* era una marca registrada.

3.2. Definición y Características

3.2.1. Arquitectura Neutra

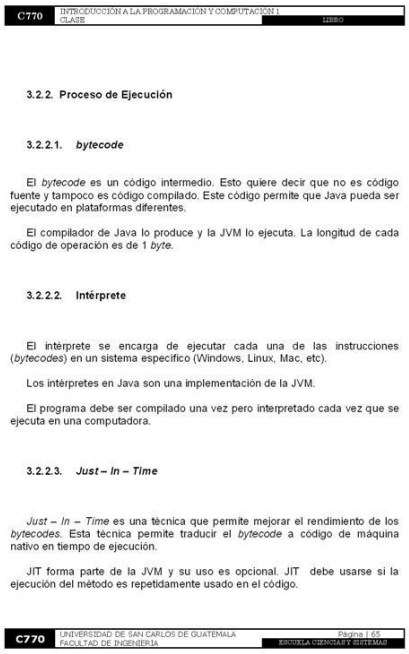
La arquitectura neutra es una característica de ciertos lenguajes que les permite ejecutarse en cualquier sistema sin importar su arquitectura. Esto lo realizan debido a que poseen un intérprete de código neutro.

3.2.1.1. *Java Virtual Machine* (JVM)

La JVM es la que se encarga de interpretar el código neutro y de obtener particularmente las instrucciones de CPU. El código neutro es código independiente del CPU y del sistema operativo.

La frase *Write Once, Run Everywhere* es el eslogan de la JVM.

Figura 26. Páginas 65 – 68 Libro “Programación principiantes”



3.2.2. Proceso de Ejecución

3.2.2.1. *bytecode*

El *bytecode* es un código intermedio. Esto quiere decir que no es código fuente y tampoco es código compilado. Este código permite que Java pueda ser ejecutado en plataformas diferentes.

El compilador de Java lo produce y la JVM lo ejecuta. La longitud de cada código de operación es de 1 *byte*.

3.2.2.2. Intérprete

El intérprete se encarga de ejecutar cada una de las instrucciones (*bytecodes*) en un sistema especifico (Windows, Linux, Mac, etc).

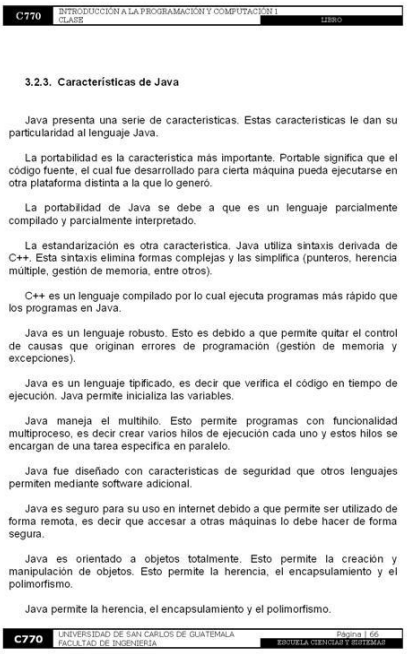
Los intérpretes en Java son una implementación de la JVM.

El programa debe ser compilado una vez pero interpretado cada vez que se ejecuta en una computadora.

3.2.2.3. *Just – In – Time*

Just – In – Time es una técnica que permite mejorar el rendimiento de los *bytecodes*. Esta técnica permite traducir el *bytecode* a código de máquina nativo en tiempo de ejecución.

JIT forma parte de la JVM y su uso es opcional. JIT debe usarse si la ejecución del método es repetidamente usado en el código.



3.2.3. Características de Java

Java presenta una serie de caracteristicas. Estas caracteristicas le dan su particularidad al lenguaje Java.

La portabilidad es la característica más importante. Portable significa que el código fuente, el cual fue desarrollado para cierta máquina pueda ejecutarse en otra plataforma distinta a la que lo generó.

La portabilidad de Java se debe a que es un lenguaje parcialmente compilado y parcialmente interpretado.

La estandarización es otra característica. Java utiliza sintaxis derivada de C++. Esta sintaxis elimina formas complejas y las simplifica (punteros, herencia múltiple, gestión de memoria, entre otros).

C++ es un lenguaje compilado por lo cual ejecuta programas más rápido que los programas en Java.

Java es un lenguaje robusto. Esto es debido a que permite quitar el control de causas que originan errores de programación (gestión de memoria y excepciones).

Java es un lenguaje tipificado, es decir que verifica el código en tiempo de ejecución. Java permite inicializa las variables.

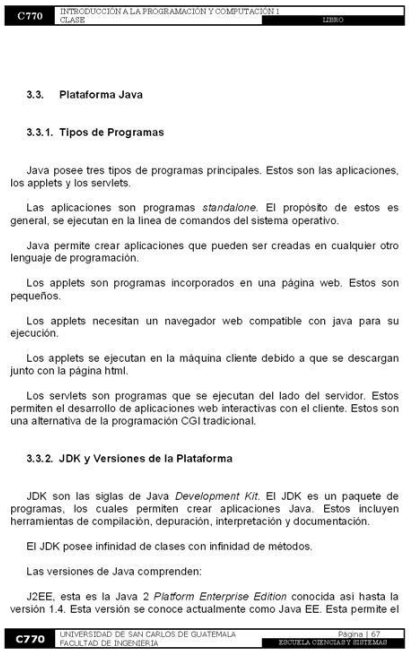
Java maneja el multihilo. Esto permite programas con funcionalidad multiproceso, es decir crear varios hilos de ejecución cada uno y estos hilos se encargan de una tarea especifica en paralelo.

Java fue diseñado con características de seguridad que otros lenguajes permiten mediante software adicional.

Java es seguro para su uso en internet debido a que permite ser utilizado de forma remota, es decir que accesar a otras máquinas lo debe hacer de forma segura.

Java es orientado a objetos totalmente. Esto permite la creación y manipulación de objetos. Esto permite la herencia, el encapsulamiento y el polimorfismo.

Java permite la herencia, el encapsulamiento y el polimorfismo.



3.3. Plataforma Java

3.3.1. Tipos de Programas

Java posee tres tipos de programas principales. Estos son las aplicaciones, los applets y los servlets.

Las aplicaciones son programas *standalone*. El propósito de estos es general, se ejecutan en la línea de comandos del sistema operativo.

Java permite crear aplicaciones que pueden ser creadas en cualquier otro lenguaje de programación.

Los applets son programas incorporados en una página web. Estos son pequeños.

Los applets necesitan un navegador web compatible con java para su ejecución.

Los applets se ejecutan en la máquina cliente debido a que se descargan junto con la página html.

Los servlets son programas que se ejecutan del lado del servidor. Estos permiten el desarrollo de aplicaciones web interactivas con el cliente. Estos son una alternativa de la programación CGI tradicional.

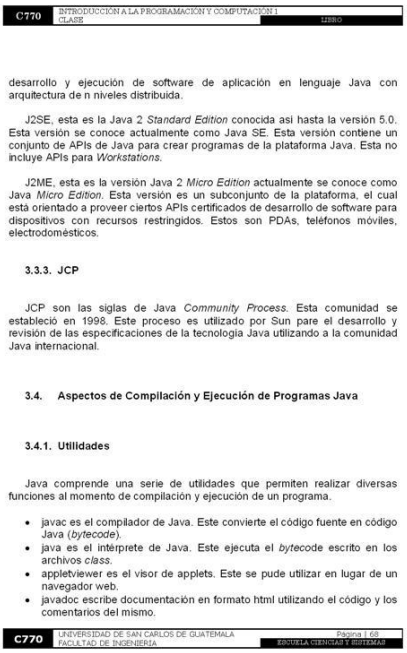
3.3.2. JDK y Versiones de la Plataforma

JDK son las siglas de Java *Development Kit*. El JDK es un paquete de programas, los cuales permiten crear aplicaciones Java. Estos incluyen herramientas de compilación, depuración, interpretación y documentación.

El JDK posee infinidad de clases con infinidad de métodos.

Las versiones de Java comprenden:

J2EE, esta es la Java 2 *Platform Enterprise Edition* conocida así hasta la versión 1.4. Esta versión se conoce actualmente como Java EE. Esta permite el



desarrollo y ejecución de software de aplicación en lenguaje Java con arquitectura de n niveles distribuida.

J2SE, esta es la Java 2 *Standard Edition* conocida así hasta la versión 5.0. Esta versión se conoce actualmente como Java SE. Esta versión contiene un conjunto de APIs de Java para crear programas de la plataforma Java. Esta no incluye APIs para *Workstations*.

J2ME, esta es la versión Java 2 *Micro Edition* actualmente se conoce como Java *Micro Edition*. Esta versión es un subconjunto de la plataforma, el cual está orientado a proveer ciertos APIs certificados de desarrollo de software para dispositivos con recursos restringidos. Estos son PDAs, teléfonos móviles, electrodomésticos.

3.3.3. JCP

JCP son las siglas de Java *Community Process*. Esta comunidad se estableció en 1998. Este proceso es utilizado por Sun pare el desarrollo y revisión de las especificaciones de la tecnologia Java utilizando a la comunidad Java internacional.

3.4. Aspectos de Compilación y Ejecución de Programas Java

3.4.1. Utilidades

Java comprende una serie de utilidades que permiten realizar diversas funciones al momento de compilación y ejecución de un programa.

- javac es el compilador de Java. Este convierte el código fuente en código Java (*bytecode*).
- java es el intérprete de Java. Este ejecuta el *bytecode* escrito en los archivos *class*.
- appletviewer es el visor de applets. Este se pude utilizar en lugar de un navegador web.
- javadoc escribe documentación en formato html utilizando el código y los comentarios del mismo.

Figura 27. Páginas 69 – 72 Libro “Programación principiantes”

- javap es el ensamblador de java.
- jar es la herramienta utilizada para trabajar con archivos rar.

3.4.2. Configuración

La configuración de Java debe seguir los siguientes pasos:

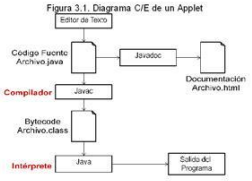
- Se debe tener instalado el JDK.
- Se debe actualizar la variable de ambiente PATH. El PATH indica al sistema operativo donde encontrar las utilidades de Java.
- Se debe crear la variable CLASSPATH.

3.4.3. Diagramas

Java presenta dos tipos de diagramas que se refieren a la compilación y ejecución.

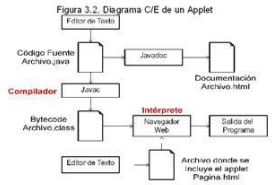
La figura 3.1. presenta el diagrama de compilación/ejecución de una aplicación en Java.

Figura 3.1. Diagrama C/E de un Applet



La figura 3.2. presenta el diagrama de compilación/ejecución de un applet en Java.

Figura 3.2. Diagrama C/E de un Applet



3.5. Estructura General de un Programa de Aplicación Java

Un programa en Java debe contener:

- Referencia a las clases que permitirán el uso de métodos (librerías).
- Una clase principal, que es un método main dentro de la clase principal.
- El nombre del archivo debe ser con extensión .java y llamarse igual que la clase principal.

La sintaxis sería:

- *import* java.io*;
  public *class* Nombre
  {
  public *static void main* (*String* [] ar)
  {
  ...//instrucciones
  }
  }

Las palabras reservadas son:

- *import*: permite la declaración que conlleva la importación de clases desde paquetes.
- public *static void main*(): método que permite la ejecución, Java exige la sintaxis.
- *static* tipo func1(...): declaración de otros métodos de la clase.

Las clases en java se agrupan en *package*. Estos *package* se encuentran en una carpeta con el mismo nombre del paquete. El *import* se utiliza para emplear un paquete dentro de un programa. La sintaxis es:

- *import* nombrePaquete.nombreClase;

Un programa es una colección de clases. Esto por lo tanto debe poseer al menos una principal (con el método *main*). Esto se declaran empezando por el acceso (opcional), luego *class*, el nombre de la clase y sus miembros.

4. ELEMENTOS BÁSICOS DE PROGRAMACIÓN Y PROGRAMACIÓN ESTRUCTURADA

4.1. Tipos de Datos

4.1.1. Tipos Primitivos

Los tipos primitivos comprenden los tipos enteros, números de coma flotante, caracteres (*char*), lógicos (*boolean*).

Los enteros permiten la representación de números tanto positivos como negativos con intervalos de valores distintos. Estos valores se listan a continuación:

- *byte*: -128 a 127.
- *short*: -32768 a 32767.
- *int*: -2147483648 a 2147483647.
- *long*: -922117036854775808 a 922117036854775807.

Los números de coma flotante son los números reales. Estos números permiten guardar valores con decimales con distinta precisión. Estos valores se listan a continuación:

- *float*: ±3.40282347e+38 a ±1.40239846e-45.
- *double*: ±1.79769313486231570e+308 a ±4.9065645841246544e-324.

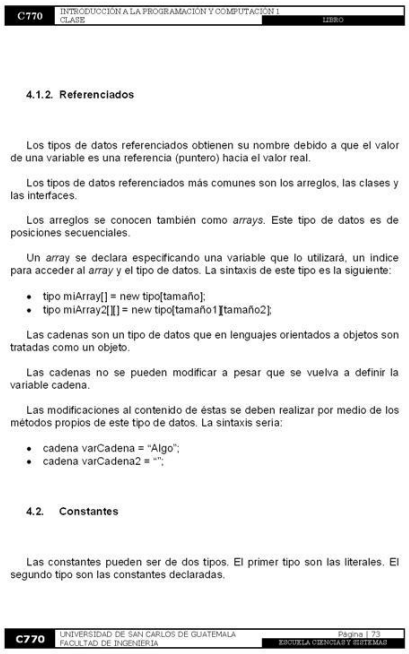
Los caracteres se conocen como *char*. Estos permiten la representación de cualquier carácter Unicode.

- *char*: \u0000 a \uFFFF.

Los lógicos se conocen también como *boolean*. Estos indican valor lógico. Los tipos lógicos poseen solamente dos valores que son: verdadero y falso.

- *boolean*: true ó false.

Figura 28. Páginas 73 – 76 Libro “Programación principiantes”



4.1.2. Referenciados

Los tipos de datos referenciados obtienen su nombre debido a que el valor de una variable es una referencia (puntero) hacia el valor real.

Los tipos de datos referenciados más comunes son los arreglos, las clases y las interfaces.

Los arreglos se conocen también como *arrays*. Este tipo de datos es de posiciones secuenciales.

Un *array* se declara especificando una variable que lo utilizará, un indice para acceder al *array* y el tipo de datos. La sintaxis de este tipo es la siguiente:

- tipo miArray[] = new tipo[tamaño];
- tipo miArray2[][] = new tipo[tamaño1][tamaño2];

Las cadenas son un tipo de datos que en lenguajes orientados a objetos son tratadas como un objeto.

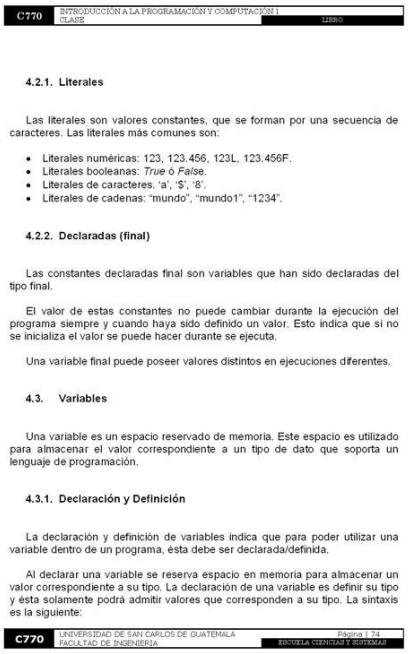
Las cadenas no se pueden modificar a pesar que se vuelva a definir la variable cadena.

Las modificaciones al contenido de éstas se deben realizar por medio de los métodos propios de este tipo de datos. La sintaxis sería:

- cadena varCadena = “Algo”;
- cadena varCadena2 = “”;

4.2. Constantes

Las constantes pueden ser de dos tipos. El primer tipo son las literales. El segundo tipo son las constantes declaradas.



4.2.1. Literales

Las literales son valores constantes, que se forman por una secuencia de caracteres. Las literales más comunes son:

- Literales numéricas: 123, 123.456, 123L, 123.456F.
- Literales booleanas: *True* ó *False*.
- Literales de caracteres, ‘a’, ‘$’, ‘8’.
- Literales de cadenas: “mundo”, “mundo1”, “1234”.

4.2.2. Declaradas (final)

Las constantes declaradas final son variables que han sido declaradas del tipo final.

El valor de estas constantes no puede cambiar durante la ejecución del programa siempre y cuando haya sido definido un valor. Esto indica que si no se inicializa el valor se puede hacer durante se ejecuta.

Una variable final puede poseer valores distintos en ejecuciones diferentes.

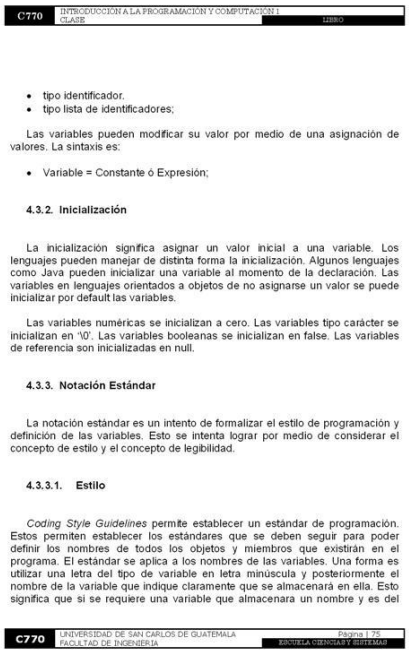
4.3. Variables

Una variable es un espacio reservado de memoria. Este espacio es utilizado para almacenar el valor correspondiente a un tipo de dato que soporta un lenguaje de programación.

4.3.1. Declaración y Definición

La declaración y definición de variables indica que para poder utilizar una variable dentro de un programa, ésta debe ser declarada/definida.

Al declarar una variable se reserva espacio en memoria para almacenar un valor correspondiente a su tipo. La declaración de una variable es definir su tipo y ésta solamente podrá admitir valores que corresponden a su tipo. La sintaxis es la siguiente:



- tipo identificador.
- tipo lista de identificadores;

Las variables pueden modificar su valor por medio de una asignación de valores. La sintaxis es:

- Variable = Constante ó Expresión;

4.3.2. Inicialización

La inicialización significa asignar un valor inicial a una variable. Los lenguajes pueden manejar de distinta forma la inicialización. Algunos lenguajes como Java pueden inicializar una variable al momento de la declaración. Las variables en lenguajes orientados a objetos de no asignarse un valor se puede inicializar por default las variables.

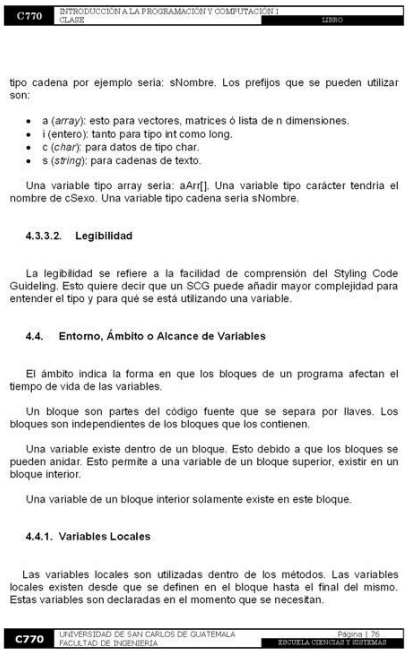
Las variables numéricas se inicializan a cero. Las variables tipo carácter se inicializan en \0’. Las variables booleanas se inicializan en false. Las variables de referencia son inicializadas en null.

4.3.3. Notación Estándar

La notación estándar es un intento de formalizar el estilo de programación y definición de las variables. Esto se intenta lograr por medio de considerar el concepto de estilo y el concepto de legibilidad.

4.3.3.1. Estilo

Coding Style Guidelines permite establecer un estándar de programación. Estos permiten establecer los estándares que se deben seguir para poder definir los nombres de todos los objetos y miembros que existirán en el programa. El estándar se aplica a los nombres de las variables. Una forma es utilizar una letra del tipo de variable en letra minúscula y posteriormente el nombre de la variable que indique claramente que se almacenará en ella. Esto significa que si se requiere una variable que almacenara un nombre y es del



tipo cadena por ejemplo sería: sNombre. Los prefijos que se pueden utilizar son:

- a (*array*): esto para vectores, matrices ó lista de n dimensiones.
- i (entero): tanto para tipo int como long.
- c (*char*): para datos de tipo char.
- s (*string*): para cadenas de texto.

Una variable tipo array sería: aArr[]. Una variable tipo carácter tendría el nombre de cSexo. Una variable tipo cadena sería sNombre.

4.3.3.2. Legibilidad

La legibilidad se refiere a la facilidad de comprensión del Styling Code Guideling. Esto quiere decir que un SCG puede añadir mayor complejidad para entender el tipo y para qué se está utilizando una variable.

4.4. Entorno, Ámbito o Alcance de Variables

El ámbito indica la forma en que los bloques de un programa afectan el tiempo de vida de las variables.

Un bloque son partes del código fuente que se separa por llaves. Los bloques son independientes de los bloques que los contienen.

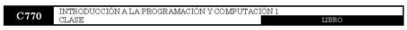
Una variable existe dentro de un bloque. Esto debido a que los bloques se pueden anidar. Esto permite a una variable de un bloque superior, existir en un bloque interior.

Una variable de un bloque interior solamente existe en este bloque.

4.4.1. Variables Locales

Las variables locales son utilizadas dentro de los métodos. Las variables locales existen desde que se definen en el bloque hasta el final del mismo. Estas variables son declaradas en el momento que se necesitan.

Figura 29. Páginas 77 – 80 Libro “Programación principiantes”



El ámbito de las variables locales se conoce como ámbito del método.

Un método definido como método_ejemplo, posee dos variables definidas como variable1 y variable2. Estas variables solamente existen dentro del método método_ejemplo.

Un segundo método es definido como método_ejemplo2. Las variables variable1 y variable2 no pueden ser utilizadas dentro del método_ejemplo2. Esto es debido a que estas variables fueron definidas localmente.

4.4.2. Ámbito de Clase

Las variables de módulos se puede decir que son variables de clase. Estas variables pueden ser utilizadas por todos los objetos de una determinada clase. Estas variables pueden ser diferentes para cada objeto. Las variables de instancia son variables de clase con la diferencia que éstas variables mantendrán su valor para todos los objetos. Estas se declaran de tipo final.

El ámbito de estos dos tipos de variables se conoce como ámbito de clase. Una clase definida como MiClase. Esta clase posee dos métodos definidos como metodo1 y metodo2. La clase también posee una variable definida como miVariable. Esta variable puede ser utilizada tanto en método1 como en metodo2.

4.5. Adicional

4.5.1. Java y C#

4.5.1.1. void

La palabra void también hace referencia a las formas de String vacias.

La palabra void se utiliza para indicar que un método no devuelve un valor. Los constructores no utilizan void, aunque ellos no devuelven valor.

- blank: x.length() != 0 && x.trim().length() == 0; en este caso “ ”.
- empty: x.length() == 0; en este caso “”.
- null: x == null

4.5.2. Java

4.5.2.1. Entrada y Salida de Datos

Java maneja la entrada y salida de datos utilizando el *System.in* y el *System.out*.

System.in es la entrada estándar. Esta proporciona un *stream* para leer del dispositivo estándar de entrada, generalmente el teclado. Este proporciona dos métodos. Estos son:

- *read*(): este deja leer un *byte* de la entrada..
- *skip*(*long* n): permite brincar n *bytes* de la entrada.

La sintaxis es:

- *System.in.read*(…)
- *Sytem.in.skip*(…)

System.out es la salida estándar. Este proporciona un *stream* para mostrar información en el dispositivo estándar de salida, generalmente la pantalla. *System.out* posee acceso a la clase *PrintStream*.

La clase *PrintStream* posee 3 métodos de visualización de información:
- *print*() y *println*(): estas dos presentan una lista de argumentos. *println*() añade además un carácter de linea nueva en el *stream*.
- *write*(): esta escribe bytes en el *stream*. Datos que no corresponden a texto.

La sintaxis es:

- *System.out.print*(…);
- *System.out.println*(…);
- *System.out.write*(…);

4.5.2.2. Tipos Básicos de Datos

Los tipos básicos que puede manejar Java son *intenger, long, float* y *double*.

La clase *Integer* maneja lo concerniente al tipo de datos *integer*. El tipo de datos *int* posee el objeto *Integer*. La sintaxis es: *Integer* I;. Esta clase brinda los siguientes métodos:

- *String Integer.toString*(*int* i, *int* base): permite convertir el entero a una cadena.
- *Int* I.parseInt (*String* s, *int* base): permite convertir una cadena a un valor entero.

La clase *Long* maneja lo concerniente al tipo de datos *Long*. El tipo de datos *long* posee el objeto *Long*. La sintaxis es: *Long* L;. Esta clase brinda los siguientes métodos:

- *String Long.toString*(*long* l, *int* base): permite convertir el *long* a una cadena.
- *long L.parseLong*(*String* s, *int* base): permite convertir una cadena a un valor long.

La clase *Float* maneja lo concerniente al tipo de datos *Float*. El tipo de datos *float* posee el objeto *Float*. La sintaxis es: *Float* F;. Esta clase brinda los siguientes métodos:

- *String Float.toString*(*float* f): permite convertir el *float* a una cadena.
- *float Float.valueOf* (*String* s): permite convertir una cadena a un valor *float*.

La clase *Double* maneja lo concerniente al tipo de datos *Double*. El tipo de datos *double* posee el objeto *Double*. La sintaxis es: *Double* D;. Esta clase brinda los siguientes métodos:

- *String Double.toString*(*double* d): permite convertir el *double* a una cadena.
- *double Double.valueOf*(*String* s): permite convertir una cadena a un valor *double*.

4.6. Expresiones y Operadores

Una expresión es una combinación de operadores y operandos, los cuales brindan un resultado particular al ser evaluados.

Los operando pueden ser variables, constantes o llamadas a métodos.

Un operador es una especie de *token* que permite indicar una evaluación o computación para ser realizada sobre objetos o datos.

4.6.1. Clasificación por Operandos

Los operadores pueden ser unarios, binarios y ternarios.

4.6.1.1. Operadores Unarios

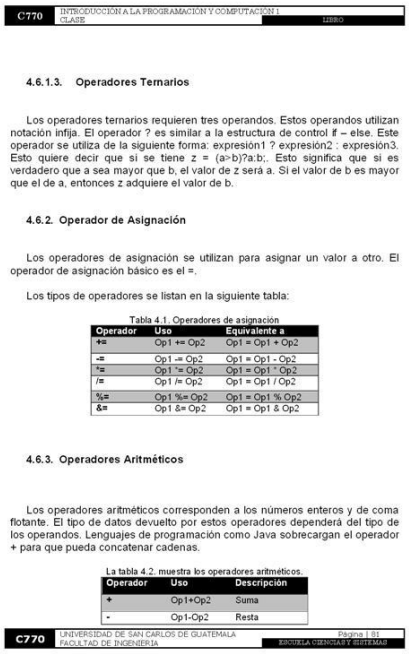
Los operadores unarios requieren solamente un operando. Estos operandos utilizan notación prefija como notación posfija. El operador clásico es el ++. Este operador unario incrementa el valor del operando en 1.

El ejemplo clásico es una variable definida entera como miEntero = 0. Este operador se utiliza de esta forma: miEntero++;. Esto permite que miEntero adquiera cada vez que se utiliza el valor de 1, 2, 3, …, n.

4.6.1.2. Operadores Binarios

Los operadores binarios requieren de dos operandos. Estos operandos utilizan notación infija. El operador = es un operador binario que permite asignar un valor del operando derecho al operando del lado izquierdo. El ejemplo permite que sea definida una variable como int miEntero. Este operador se utiliza colocando: miEntero = 2;. Esto ocasiona que la variable adquiera el valor de 2.

Figura 30. Páginas 81 – 84 Libro “Programación principiantes”



4.6.1.3. Operadores Ternarios

Los operadores ternarios requieren tres operandos. Estos operandos utilizan notación infija. El operador ? es similar a la estructura de control if – else. Este operador se utiliza de la siguiente forma: expresión1 ? expresión2 : expresión3. Esto quiere decir que si se tiene z = (a>b)?a:b;. Esto significa que si es verdadero que a sea mayor que b, el valor de z será a. Si el valor de b es mayor que el de a, entonces z adquiere el valor de b.

4.6.2. Operador de Asignación

Los operadores de asignación se utilizan para asignar un valor a otro. El operador de asignación básico es el =.

Los tipos de operadores se listan en la siguiente tabla:

Tabla 4.1. Operadores de asignación

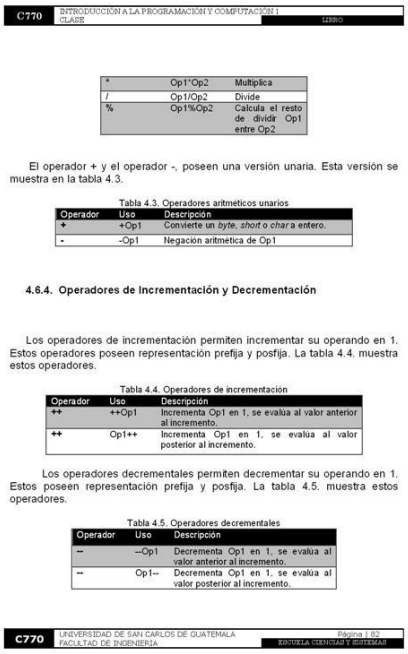
Operador	Uso	Equivalente a
+=	Op1 += Op2	Op1 = Op1 + Op2
-=	Op1 -= Op2	Op1 = Op1 - Op2
*=	Op1 *= Op2	Op1 = Op1 * Op2
/=	Op1 /= Op2	Op1 = Op1 / Op2
%=	Op1 %= Op2	Op1 = Op1 % Op2
&=	Op1 &= Op2	Op1 = Op1 & Op2

4.6.3. Operadores Aritméticos

Los operadores aritméticos corresponden a los números enteros y de coma flotante. El tipo de datos devuelto por estos operadores dependerá del tipo de los operandos. Lenguajes de programación como Java sobrecargan el operador + para que pueda concatenar cadenas.

La tabla 4.2. muestra los operadores aritméticos.

Operador	Uso	Descripción
+	Op1+Op2	Suma
-	Op1-Op2	Resta



*	Op1*Op2	Multiplica
/	Op1/Op2	Divide
%	Op1%Op2	Calcula el resto de dividir Op1 entre Op2

El operador + y el operador -, poseen una versión unaria. Esta versión se muestra en la tabla 4.3.

Tabla 4.3. Operadores aritméticos unarios

Operador	Uso	Descripción
+	+Op1	Convierte un *byte*, *short* o *char* a entero.
-	-Op1	Negación aritmética de Op1

4.6.4. Operadores de Incrementación y Decrementación

Los operadores de incrementación permiten incrementar su operando en 1. Estos operadores poseen representación prefija y posfija. La tabla 4.4. muestra estos operadores.

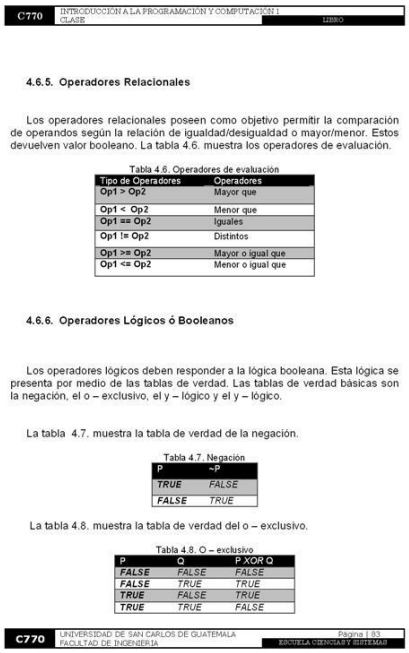
Tabla 4.4. Operadores de incrementación

Operador	Uso	Descripción
++	++Op1	Incrementa Op1 en 1, se evalúa al valor anterior al incremento.
++	Op1++	Incrementa Op1 en 1, se evalúa al valor posterior al incremento.

Los operadores decrementales permiten decrementar su operando en 1. Estos poseen representación prefija y posfija. La tabla 4.5. muestra estos operadores.

Tabla 4.5. Operadores decrementales

Operador	Uso	Descripción
--	--Op1	Decrementa Op1 en 1, se evalúa al valor anterior al incremento.
--	Op1--	Decrementa Op1 en 1, se evalúa al valor posterior al incremento.



4.6.5. Operadores Relacionales

Los operadores relacionales poseen como objetivo permitir la comparación de operandos según la relación de igualdad/desigualdad o mayor/menor. Estos devuelven valor booleano. La tabla 4.6. muestra los operadores de evaluación.

Tabla 4.6. Operadores de evaluación

Tipo de Operadores	Operadores
Op1 > Op2	Mayor que
Op1 < Op2	Menor que
Op1 == Op2	Iguales
Op1 != Op2	Distintos
Op1 >= Op2	Mayor o igual que
Op1 <= Op2	Menor o igual que

4.6.6. Operadores Lógicos ó Booleanos

Los operadores lógicos deben responder a la lógica booleana. Esta lógica se presenta por medio de las tablas de verdad. Las tablas de verdad básicas son la negación, el o – exclusivo, el y – lógico y el y – lógico.

La tabla 4.7. muestra la tabla de verdad de la negación.

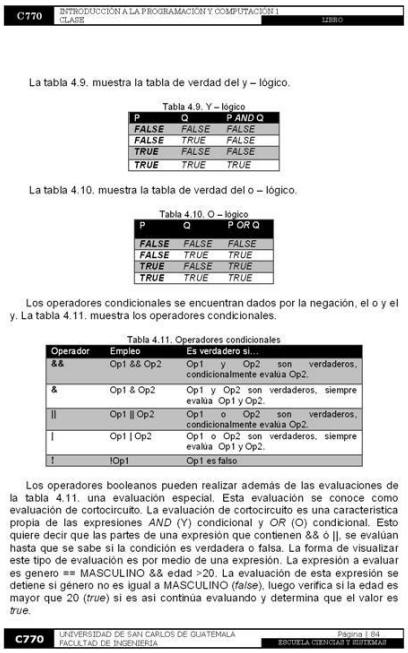
Tabla 4.7. Negación

P	~P
TRUE	*FALSE*
FALSE	*TRUE*

La tabla 4.8. muestra la tabla de verdad del o – exclusivo.

Tabla 4.8. O – exclusivo

P	Q	P XOR Q
FALSE	*FALSE*	*FALSE*
FALSE	*TRUE*	*TRUE*
TRUE	*FALSE*	*TRUE*
TRUE	*TRUE*	*FALSE*



La tabla 4.9. muestra la tabla de verdad del y – lógico.

Tabla 4.9. Y – lógico

P	Q	P AND Q
FALSE	*FALSE*	*FALSE*
FALSE	*TRUE*	*FALSE*
TRUE	*FALSE*	*FALSE*
TRUE	*TRUE*	*TRUE*

La tabla 4.10. muestra la tabla de verdad del o – lógico.

Tabla 4.10. O – lógico

P	Q	P OR Q
FALSE	*FALSE*	*FALSE*
FALSE	*TRUE*	*TRUE*
TRUE	*FALSE*	*TRUE*
TRUE	*TRUE*	*TRUE*

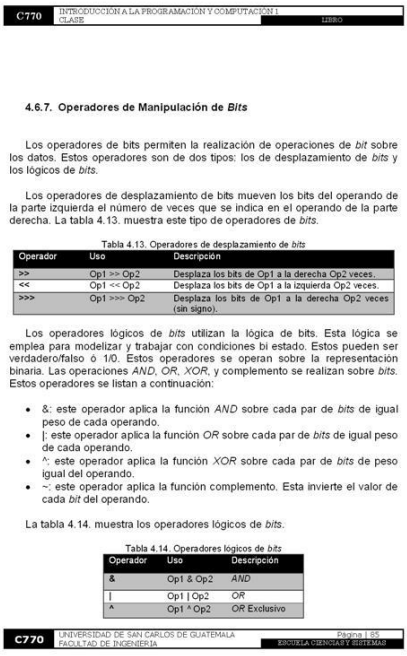
Los operadores condicionales se encuentran dados por la negación, el o y el y. La tabla 4.11. muestra los operadores condicionales.

Tabla 4.11. Operadores condicionales

Operador	Empleo	Es verdadero si...
&&	Op1 && Op2	Op1 y Op2 son verdaderos, condicionalmente evalúa Op2.
&	Op1 & Op2	Op1 y Op2 son verdaderos, siempre evalúa Op1 y Op2.
||	Op1 || Op2	Op1 o Op2 son verdaderos, condicionalmente evalúa Op2.
|	Op1 | Op2	Op1 o Op2 son verdaderos, siempre evalúa Op1 y Op2.
!	!Op1	Op1 es falso

Los operadores booleanos pueden realizar además de las evaluaciones de la tabla 4.11. una evaluación especial. Esta evaluación se conoce como evaluación de cortocircuito. La evaluación de cortocircuito es una característica propia de las expresiones *AND* (Y) condicional y *OR* (O) condicional. Esto quiere decir que las partes de una expresión que contienen && ó ||, se evalúan hasta que se sabe si la condición es verdadera o falsa. La forma de visualizar este tipo de evaluación es por medio de una expresión. La expresión a evaluar es genero == MASCULINO && edad >20. La evaluación de esta expresión se detiene si género no es igual a MASCULINO (*false*), luego verifica si la edad es mayor que 20 (*true*) si es así continúa evaluando y determina que el valor es *true*.

Figura 31. Páginas 85 – 88 Libro “Programación principiantes”



4.6.7. Operadores de Manipulación de *Bits*

Los operadores de bits permiten la realización de operaciones de *bit* sobre los datos. Estos operadores son de dos tipos: los de desplazamiento de *bits* y los lógicos de *bits*.

Los operadores de desplazamiento de bits mueven los bits del operando de la parte izquierda el número de veces que se indica en el operando de la parte derecha. La tabla 4.13. muestra este tipo de operadores de *bits*.

Tabla 4.13. Operadores de desplazamiento de *bits*

Operador	Uso	Descripción
>>	Op1 >> Op2	Desplaza los bits de Op1 a la derecha Op2 veces.
<<	Op1 << Op2	Desplaza los bits de Op1 a la izquierda Op2 veces.
>>>	Op1 >>> Op2	Desplaza los bits de Op1 a la derecha Op2 veces (sin signo).

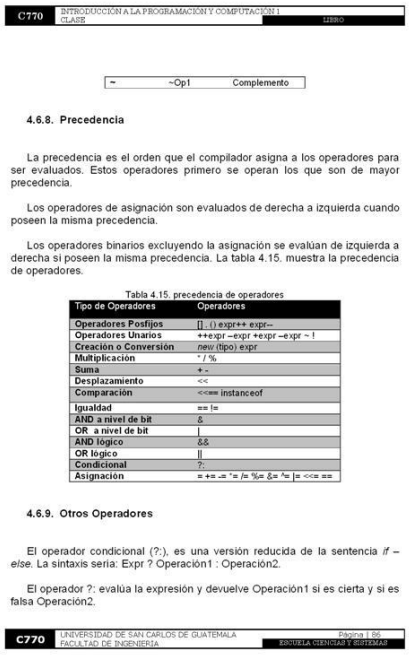
Los operadores lógicos de *bits* utilizan la lógica de bits. Esta lógica se emplea para modelizar y trabajar con condiciones bi estado. Estos pueden ser verdadero/falso ó 1/0. Estos operadores se operan sobre la representación binaria. Las operaciones *AND*, *OR*, *XOR*, y complemento se realizan sobre *bits*. Estos operadores se listan a continuación:

- &: este operador aplica la función *AND* sobre cada par de *bits* de igual peso de cada operando.
- |: este operador aplica la función *OR* sobre cada par de *bits* de igual peso de cada operando.
- ^: este operador aplica la función *XOR* sobre cada par de *bits* de peso igual del operando.
- ~: este operador aplica la función complemento. Esta invierte el valor de cada *bit* del operando.

La tabla 4.14. muestra los operadores lógicos de *bits*.

Tabla 4.14. Operadores lógicos de *bits*

Operador	Uso	Descripción
&	Op1 & Op2	*AND*
|	Op1 | Op2	*OR*
^	Op1 ^ Op2	*OR* Exclusivo



~	~Op1	Complemento

4.6.8. Precedencia

La precedencia es el orden que el compilador asigna a los operadores para ser evaluados. Estos operadores primero se operan los que son de mayor precedencia.

Los operadores de asignación son evaluados de derecha a izquierda cuando poseen la misma precedencia.

Los operadores binarios excluyendo la asignación se evalúan de izquierda a derecha si poseen la misma precedencia. La tabla 4.15. muestra la precedencia de operadores.

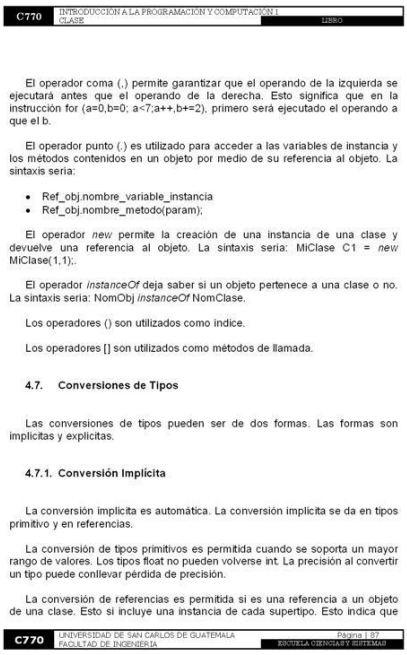
Tabla 4.15. precedencia de operadores

Tipo de Operadores	Operadores
Operadores Posfijos	[], . (), expr++ expr--
Operadores Unarios	++expr --expr +expr -expr ~ !
Creación o Conversión	*new* (tipo) expr
Multiplicación	* / %
Suma	+ -
Desplazamiento	<<
Comparación	<<= instanceof
Igualdad	== !=
AND a nivel de bit	&
OR a nivel de bit	|
AND lógico	&&
OR lógico	||
Condicional	?:
Asignación	= += -= *= /= %= &= ^= |= <<= ==

4.6.9. Otros Operadores

El operador condicional (?:), es una versión reducida de la sentencia *if – else*. La sintaxis sería: Expr ? Operación1 : Operación2.

El operador ?: evalúa la expresión y devuelve Operación1 si es cierta y si es falsa Operación2.



El operador coma (,) permite garantizar que el operando de la izquierda se ejecutará antes que el operando de la derecha. Esto significa que en la instrucción for (a=0,b=0; a<7;a++,b+=2), primero será ejecutado el operando a que el b.

El operador punto (.) es utilizado para acceder a las variables de instancia y los métodos contenidos en un objeto por medio de su referencia al objeto. La sintaxis sería:

- Ref_obj.nombre_variable_instancia
- Ref_obj.nombre_metodo(param);

El operador *new* permite la creación de una instancia de una clase y devuelve una referencia al objeto. La sintaxis sería: MiClase C1 = *new* MiClase(1,1);.

El operador *instanceOf* deja saber si un objeto pertenece a una clase o no. La sintaxis sería: NomObj *instanceOf* NomClase.

Los operadores () son utilizados como índice.

Los operadores [] son utilizados como métodos de llamada.

4.7. Conversiones de Tipos

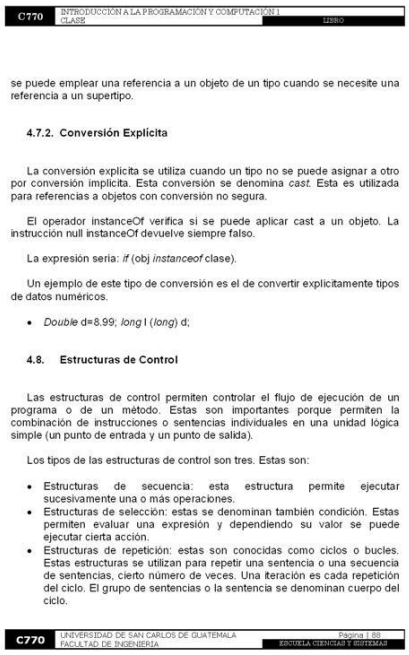
Las conversiones de tipos pueden ser de dos formas. Las formas son implícitas y explícitas.

4.7.1. Conversión Implícita

La conversión implícita es automática. La conversión implícita se da en tipos primitivo y en referencias.

La conversión de tipos primitivos es permitida cuando se soporta un mayor rango de valores. Los tipos float no pueden volverse int. La precisión al convertir un tipo puede conllevar pérdida de precisión.

La conversión de referencias es permitida si es una referencia a un objeto de una clase. Esto si incluye una instancia de cada supertipo. Esto indica que



se puede emplear una referencia a un objeto de un tipo cuando se necesite una referencia a un supertipo.

4.7.2. Conversión Explícita

La conversión explícita se utiliza cuando un tipo no se puede asignar a otro por conversión implícita. Esta conversión se denomina *cast*. Esta es utilizada para referencias a objetos con conversión no segura.

El operador instanceOf verifica si se puede aplicar cast a un objeto. La instrucción null instanceOf devuelve siempre falso.

La expresión sería: *if* (obj *instanceof* clase).

Un ejemplo de este tipo de conversión es el de convertir explícitamente tipos de datos numéricos.

- *Double* d=8.99; *long* l (*long*) d;

4.8. Estructuras de Control

Las estructuras de control permiten controlar el flujo de ejecución de un programa o de un método. Estas son importantes porque permiten la combinación de instrucciones o sentencias individuales en una unidad lógica simple (un punto de entrada y un punto de salida).

Los tipos de las estructuras de control son tres. Estas son:

- Estructuras de secuencia: esta estructura permite ejecutar sucesivamente una o más operaciones.
- Estructuras de selección: estas se denominan también condición. Estas permiten evaluar una expresión y dependiendo su valor se puede ejecutar cierta acción.
- Estructuras de repetición: estas son conocidas como ciclos o bucles. Estas estructuras se utilizan para repetir una sentencia o una secuencia de sentencias, cierto número de veces. Una iteración es cada repetición del ciclo. El grupo de sentencias o la sentencia se denominan cuerpo del ciclo.

Figura 32. Páginas 89 – 92 Libro “Programación principiantes”

4.8.1. Sentencia Compuesta

Una sentencia compuesta es un conjunto de sentencias que se encierran entre llaves ({}) que se utiliza para la especificación de un flujo secuencial. La sintaxis es:

- {
  Sentencia1;
  Sentencia2;
  ...
  SentenciaN;
  }

4.8.2. Estructuras de Control Condicionales

Las estructuras de control condicionales se pueden dividir en dos formas. Las estructuras si – sino y las estructuras en caso.

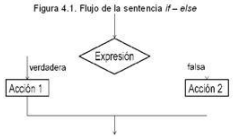
4.8.2.1. Si – Sino

Las estructuras si – sino también se conocen como *if – else*. La estructura de control *if* es la estructura de selección principal. El formato de esta estructura es:

- *if* (expresión)
  {//se realiza esta acción1 en caso de ser cierta la expresión (condición)
  }
  else
  {//en caso de ser falsa la expresión se realiza esta acción2
  }

La sentencia *if – else* obedece al diagrama de flujo de la figura 4.1.

Figura 4.1. Flujo de la sentencia *if – else*



Una sentencia *if* se considera anidada al momento de que la sentencia de la rama verdadera o falsa es otra sentencia *if*. La sintaxis sería:

- *if* (condición1)
  {
  sentencia1
  }
  else
  {
  if (condición2)
  {
  sentencia2
  ...//existen mas sentencias e *if*'s
  }
  else
  {
  if (condiciónn)
  {
  sentencian
  }
  else
  {
  sentenciam}
  }
  }

4.8.2.2. En Caso

La sentencia en caso se conoce también como *switch/case*. La sentencia *switch* es utilizada para seleccionar una entre varias opciones.

El valor del selector debe ser entero o carácter. La sintaxis de esta sentencia es:

- *switch* (selector)
  {
  case etiqueta1: setencias1;
  break;
  case etiqueta2: setencias2;
  break;
  ... //existen más etiquetas
  case etiquetan: setenciasn;
  break;
  default: sentenciasm; //es opcional
  }

El selector se refiere a una variable o una expresión tipo entero, carácter o booleano. La etiqueta indica que valor puede tomar el selector, en caso de ser tipo carácter va entre apostrofes (‘’).

La instrucción *break* permite alcanzar el final del *switch*. La instrucción *default* permite realizar una sentencia en caso el selector no coincida con ninguna etiqueta.

La evaluación de esta sentencia sigue los siguientes pasos.

- El selector es evaluado y comparado con cada una de las etiquetas de *case*.
- Las etiquetas deben de poseer un valor único, constante y el valor debe ser distinto entre las demás etiquetas.
- Al momento de que el selector coincida con alguna etiqueta, se realizan las sentencias contenidas en el *case* hasta que se encuentre la palabra *break* ó hasta que se encuentre el final de la estructura *switch*.
- Las etiquetas deben poseer el mismo tipo del selector. Las expresiones están permitidas como etiquetas solamente si la expresión en sí misma es una constante.
- Se coloca la palabra reservada *default* para indicar que en caso el valor del selector no coincida con ningún valor de las etiquetas se realice las

sentencias que existen en default. Esta se puede o no incluir en la estructura.

4.8.3. Estructuras de Control Repetitivas o Iterativas

Las estructuras de control repetitivas poseen tres partes importantes. Estas partes son:

- Bucle: este es cualquier estructura de un programa que repite una sentencia o conjunto de sentencias, una o más veces.
- Cuerpo: este es la sentencia o conjunto de sentencias que se repite.
- Iteración: este es cada repetición del cuerpo del bucle.

La construcción de un bucle debe considerar dos preguntas importantes. La primera corresponde a ¿Cómo estará constituido el cuerpo del bucle?. La segunda corresponde a ¿Cuántas veces se repetirá el cuerpo del bucle?.

Las estructuras de control repetitivas utilizan una variable de control. Esta variable permite representar la condición del bucle. El valor de esta variable permite determinar si el cuerpo del bucle debe repetirse. La variable de control posee las siguientes etapas:

- Inicialización: el contador es inicializado a 0 o a algún otro valor, antes de empezar el bucle.
- Prueba: se verifica el valor del contador para poder realizar la iteración.
- Actualización: el contador es actualizado durante cada iteración.

Las tres estructuras de repetición básicas son: mientras, para y hacer mientras.

4.8.3.1. Mientras

La estructura mientras se conoce también como *while*. El bucle while permite mientras se cumpla la condición se realizará una iteración. La condición es evaluada antes que se ejecute el cuerpo del bucle.

El bucle se encuentra formado por:

- La palabra reservada *while*.

Figura 33. Páginas 93 – 96 Libro “Programación principiantes”

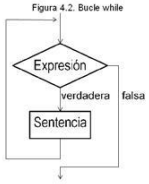
- La condición del bucle, que es la condición lógica o booleana.
- Las sentencias.

La sintaxis del bucle *while* es:

- *while* (condición_bucle)
  {
  sentencia1;
  sentencia2;
  //existen más sentencias
  sentencian;
  }

La figura 4.2. presenta el diagrama de flujo del bucle *while*.

Figura 4.2. Bucle while



Un bucle *while* se repite siempre y cuando su condición sea verdadera. Esto permite colocar la condición para que sea verdadera siempre y por medio de una sentencia *break* después de cierta condición terminar el bucle. La condición nunca será falsa.

4.8.3.2. Para

El bucle para es conocido también como *for*. El bucle *for* permite un tipo de bucle controlado por contador. Estos ejecutan el conjunto de sentencias una vez por cada valor de un rango especificado.

El bucle *for* se encuentra integrado por:

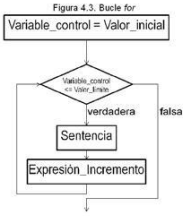
- La palabra reservada *for*.
- La inicialización.
- La condición de la iteración.
- El incremento.

La sintaxis del bucle *for* es la siguiente:

- *for* (*int* i=0; i<10; i++)
  {
  sentencia1;
  sentencia2;
  //hay mas sentencias
  sentencian;
  }

La figura 4.3. muestra el diagrama de flujo del bucle para.

Figura 4.3. Bucle *for*



Los bucles *for* pueden contener dos ocurrencias que pueden provocar algún funcionamiento anómalo en el bucle.

La primera ocurrencia corresponde a las sentencias nulas. Esto indica que una o varias de las sentencias dentro de los paréntesis de un bucle *for* pueden ser nulas. Esto se provoca por medio de colocar ";".

Un caso de esto puede ser:

- *int* contador=0;
  for (; contador<5;)
  {
  contador++;
  }

La segunda ocurrencia son los bucles vacios. Un bucle vacio puede ocurrir debido a que se colocó un ";" al terminar el ")". Este permite ejecutar el bucle el número de veces indicado pero no ejecutar nada. Esto sería:

- *for* (*int* i=1; i<10; i++);

4.8.3.3. Hacer Mientras

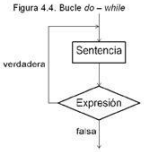
El bucle hacer mientras se conoce como *do – while*. Este es un bucle condicional que se ejecuta al menos una vez.

La sintaxis de este bucle es:

- *do*
  {
  sentencia1;
  sentencia2;
  //mas sentencias
  sentencian;
  } *while* (expresion)

La figura 4.4. presenta el diagrama de flujo del ciclo *do – while*.

Figura 4.4. Bucle *do – while*



La diferencia principal entre un bucle do – while y un bucle while, es que el bucle do – while se ejecutará al menos una vez.

4.8.4. Comparación entre los Diferentes Bucles

Los distintos bucles poseen diversas características que los diferencian los unos de los otros. La tabla 4.16. muestra esta comparativa.

Tabla 4.16. Comparativa entre los diferentes bucles

Bucle	Formato
while	Este es utilizado cuando la condición no se encuentra determinada por un contador. La evaluación de la condición precede a cada iteración. Esto permite que el bucle pueda no ser ejecutado. Generalmente se utiliza cuando se desea saltar si la condición es falsa.
for	Este permite llevar un control por medio de un contador. Esto es debido a que el número de repeticiones se conoce por anticipado. La evaluación del contador precede a la ejecución del cuerpo del bucle.
do – while	Este es utilizado al momento de que se necesite la ejecución del bucle al menos una vez. Esto debido a que la evaluación se realiza después de la ejecución del cuerpo del bucle. En todo lo demás es igual al bucle while.

4.8.5. Recomendaciones

El bucle *while* es repetido siempre y cuando la condición sea verdadera.

El bucle *for* es utilizado cuando se conoce con anticipación cuantas veces se debe repetir.

Figura 34. Páginas 97 – 100 Libro “Programación principiantes”

El bucle *do – while* es similar al *while* solamente difiere en que el cuerpo se ejecutará al menos una vez.

4.8.6. Anidación de Bucles

Los bucles se pueden anidar. Los bucles anidados poseen un bucle externo con uno o más bucles internos. El proceso es que cuando un bucle externo es repetido, los bucles internos se repiten. El bucle interno es ejecutado desde el principio hasta el final, por cada iteración del bucle externo.

4.8.7. Diseño Eficiente de Bucles

El diseño eficiente de bucles permite considerar cero iteraciones, bucles infinitos, manejo de incrementos y decrementos, terminaciones anormales de un ciclo, los modelos, las precauciones y el fin de bucles.

4.8.7.1. Cero Iteraciones

Las cero iteraciones surgen cuando un bucle no se ejecuta debido a que la condición de repetición del bucle no se cumple. Esto es para la primera vez que se ejecuta el bucle. Esto se da suponiendo que se tenga:

```
int val = 0;
while (val > 0)
{
    val = val -1;
}
```

Este bucle nunca realizará ninguna iteración debido a que su expresión siempre será falsa y el bucle no se ejecuta.

4.8.7.2. Bucles Infinitos

Los bucles infinitos son los bucles que en un principio no poseen fin. Estos para interrumpirlos se utilizan *if – break* ó *if – return*. La sintaxis de estos bucles es:

- *for* (;;){}.

El cuerpo del bucle debe contener las sentencias que se desee ejecutar repetidamente.

La ejecución de una sentencia se terminará cuando se cumpla una determinada solución. La sintaxis es:

- *if* (condición)
  {
  break;
  }

4.8.7.3. Manejo de Incrementos y Decrementos

Los operadores de incremento son ++ y de decremento son -- de una variable.

- ++nombre: preincremento.
- nombre++: postincremento.
- --nombre: predecremento.
- nombre--: postdecremento.

4.8.7.4. Terminaciones Anormales de un Ciclo

Un ciclo se encuentra determinado por la palabra reservada, los paréntesis y las llaves.

Las llaves al no ser colocadas para encerrar el cuerpo del bucle puede darse un ciclo infinito y no realizar las sentencias.

4.8.7.5. Modelos

Los modelos utilizados en los bucles pueden ser de dos tipos.

El primer tipo corresponde al modelo de control por centinelas.

El centinela es un único dato definido y especificado como último dato.

El bucle es ejecutado, lee cada dato y termina cuando se lee el valor del centinela.

El valor del centinela debe ser uno que no sea utilizado como dato. El valor del centinela terminará el bucle.

Este caso sería que se posee:

```
int centinela = -1;
int nota = 0;
int suma = 0;
leer (nota);
while (nota ¡= centinela)
{
    suma = suma + nota;
    leer(nota);
}
```

Este caso permite que únicamente se salga del bucle si se introduce el valor de nota igual al de centinela.

El valor de ser distinto sumará números de notas hasta que se coloque el valor de centinela y se incumpla la condición.

El segundo modelo es el de control por indicadores.

Una bandera es una variable de tipo booleano que permitirá controlar la ejecución de un bucle. Esta se inicializa generalmente en falso y posteriormente por una acción determinada dentro del bucle, es puesto en verdadero y esto finalizará el bucle.

El pseudocódigo siguiente muestra el control por indicadores:

```
int numero = 0;
int suma = 0;
```



```
bool bandera = true;
leer(numero)
while (bandera)
{
    suma = suma + numero;
    leer(numero);
    if (numero < 0)
        bandera = false;
}
```

La variable de control debe permitir que en algún momento la condición del bucle sea falsa. Esto indica que se debe tener cuidado de no modificar el valor de la variable de control. Esto debido a que puede que la condición no sea falsa y el bucle convertirse en un bucle infinito.

4.8.7.6. Fin de un Bucle

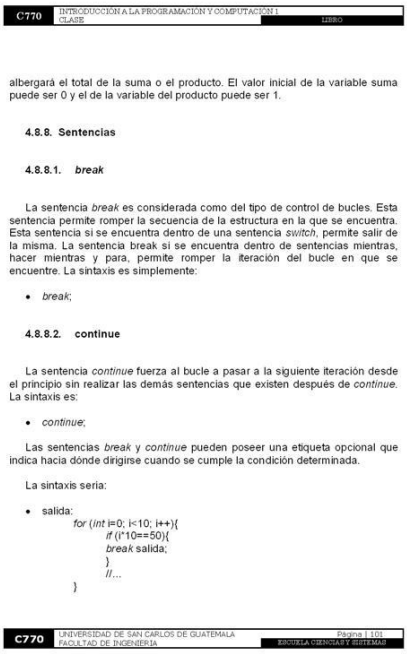
Los bucles pueden ser terminados de 4 métodos distintos. Estos métodos son:

- Lista encabezada por tamaño: esta permite conocer el tamaño de la lista ya sea por algún método o por medio del usuario. De esta forma se repetirá el número de veces que corresponde al tamaño de la lista.
- Preguntar antes de la iteración: esta indica que se debe preguntar al usuario cada vez que se termine el bucle si se desea continuar o no.
- Lista terminada con un valor centinela: esta permite por medio de un valor centinela que es un valor distinto de cualquier valor que puede adquirir la lista. Si se lee este valor distinto se termina el bucle.
- Agotamiento de la entrada: esto permite que cuando se han leído todas las entradas ya sea de un archivo u otro medio, se finaliza el bucle. Este método es el más utilizado para lectura de archivos y el valor para terminar el bucle es el fin de archivo.

4.8.7.7. Bucles para Diseño de Sumas y Productos

El diseño de una suma o un producto puede ser realizado por medio de un bucle for. El ciclo for permite realizar una suma o un producto teniendo en cuenta el total de datos a sumar. Este bucle debe de tener una variable que

Figura 35. Páginas 101 – 104 Libro “Programación principiantes”



albergará el total de la suma o el producto. El valor inicial de la variable suma puede ser 0 y el de la variable del producto puede ser 1.

4.8.8. Sentencias

4.8.8.1. *break*

La sentencia *break* es considerada como del tipo de control de bucles. Esta sentencia permite romper la secuencia de la estructura en la que se encuentra. Esta sentencia si se encuentra dentro de una sentencia *switch*, permite salir de la misma. La sentencia break si se encuentra dentro de sentencias mientras, hacer mientras y para, permite romper la iteración del bucle en que se encuentre. La sintaxis es simplemente:

- *break*;

4.8.8.2. continue

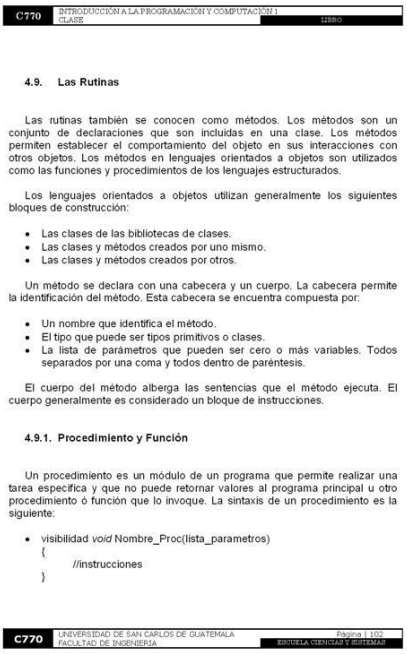
La sentencia *continue* fuerza al bucle a pasar a la siguiente iteración desde el principio sin realizar las demás sentencias que existen después de *continue*. La sintaxis es:

- *continue*;

Las sentencias *break* y *continue* pueden poseer una etiqueta opcional que indica hacia dónde dirigirse cuando se cumple la condición determinada.

La sintaxis sería:

- salida:
  for (*int* i=0; i<10; i++){
  if (i*10==50){
  break salida;
  }
  //...
  }



4.9. Las Rutinas

Las rutinas también se conocen como métodos. Los métodos son un conjunto de declaraciones que son incluidas en una clase. Los métodos permiten establecer el comportamiento del objeto en sus interacciones con otros objetos. Los métodos en lenguajes orientados a objetos son utilizados como las funciones y procedimientos de los lenguajes estructurados.

Los lenguajes orientados a objetos utilizan generalmente los siguientes bloques de construcción:

- Las clases de las bibliotecas de clases.
- Las clases y métodos creados por uno mismo.
- Las clases y métodos creados por otros.

Un método se declara con una cabecera y un cuerpo. La cabecera permite la identificación del método. Esta cabecera se encuentra compuesta por:

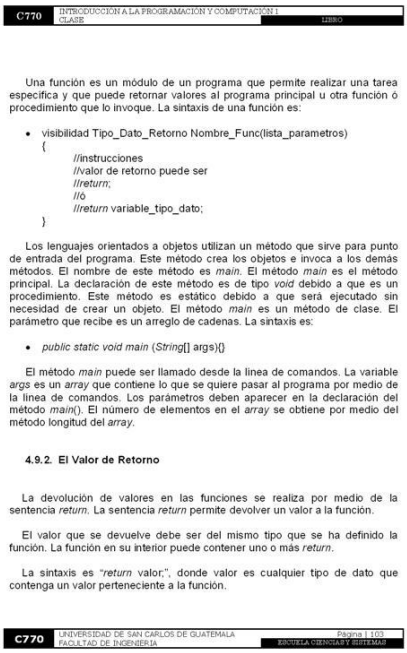
- Un nombre que identifica el método.
- El tipo que puede ser tipos primitivos o clases.
- La lista de parámetros que pueden ser cero o más variables. Todos separados por una coma y todos dentro de paréntesis.

El cuerpo del método alberga las sentencias que el método ejecuta. El cuerpo generalmente es considerado un bloque de instrucciones.

4.9.1. Procedimiento y Función

Un procedimiento es un módulo de un programa que permite realizar una tarea especifica y que no puede retornar valores al programa principal u otro procedimiento ó función que lo invoque. La sintaxis de un procedimiento es la siguiente:

- visibilidad *void* Nombre_Proc(lista_parametros)
  {
  //instrucciones
  }



Una función es un módulo de un programa que permite realizar una tarea especifica y que puede retornar valores al programa principal u otra función ó procedimiento que lo invoque. La sintaxis de una función es:

- visibilidad Tipo_Dato_Retorno Nombre_Func(lista_parametros)
  {
  //instrucciones
  //valor de retorno puede ser
  //*return*;
  //ó
  //*return* variable_tipo_dato;
  }

Los lenguajes orientados a objetos utilizan un método que sirve para punto de entrada del programa. Este método crea los objetos e invoca a los demás métodos. El nombre de este método es *main*. El método *main* es el método principal. La declaración de este método es de tipo *void* debido a que es un procedimiento. Este método es estático debido a que será ejecutado sin necesidad de crear un objeto. El método *main* es un método de clase. El parámetro que recibe es un arreglo de cadenas. La sintaxis es:

- *public static void main* (*String*[] args){}

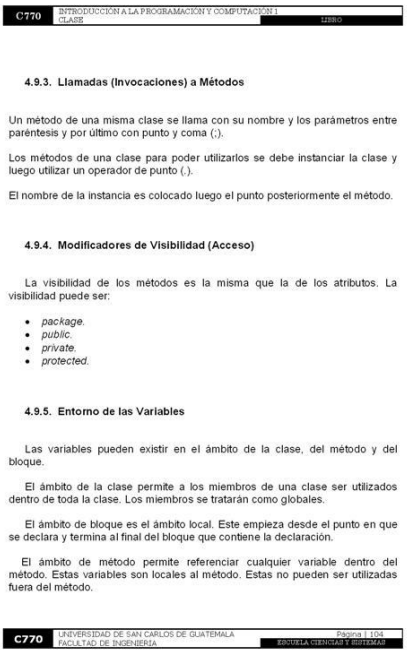
El método *main* puede ser llamado desde la línea de comandos. La variable *args* es un *array* que contiene lo que se quiere pasar al programa por medio de la línea de comandos. Los parámetros deben aparecer en la declaración del método *main*(). El número de elementos en el *array* se obtiene por medio del método longitud del *array*.

4.9.2. El Valor de Retorno

La devolución de valores en las funciones se realiza por medio de la sentencia *return*. La sentencia *return* permite devolver un valor a la función.

El valor que se devuelve debe ser del mismo tipo que se ha definido la función. La función en su interior puede contener uno o más *return*.

La sintaxis es “*return* valor;”, donde valor es cualquier tipo de dato que contenga un valor perteneciente a la función.



4.9.3. Llamadas (Invocaciones) a Métodos

Un método de una misma clase se llama con su nombre y los parámetros entre paréntesis y por último con punto y coma (;).

Los métodos de una clase para poder utilizarlos se debe instanciar la clase y luego utilizar un operador de punto (.).

El nombre de la instancia es colocado luego el punto posteriormente el método.

4.9.4. Modificadores de Visibilidad (Acceso)

La visibilidad de los métodos es la misma que la de los atributos. La visibilidad puede ser:

- *package*.
- *public*.
- *private*.
- *protected*.

4.9.5. Entorno de las Variables

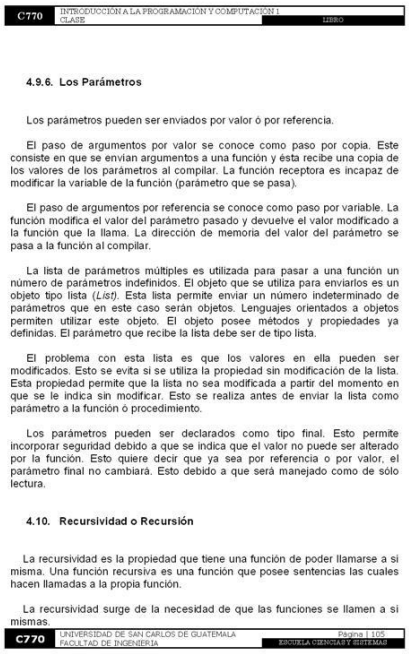
Las variables pueden existir en el ámbito de la clase, del método y del bloque.

El ámbito de la clase permite a los miembros de una clase ser utilizados dentro de toda la clase. Los miembros se tratarán como globales.

El ámbito de bloque es el ámbito local. Este empieza desde el punto en que se declara y termina al final del bloque que contiene la declaración.

El ámbito de método permite referenciar cualquier variable dentro del método. Estas variables son locales al método. Estas no pueden ser utilizadas fuera del método.

Figura 36. Páginas 105 – 108 Libro “Programación principiantes”



4.9.6. Los Parámetros

Los parámetros pueden ser enviados por valor ó por referencia.

El paso de argumentos por valor se conoce como paso por copia. Este consiste en que se envían argumentos a una función y ésta recibe una copia de los valores de los parámetros al compilar. La función receptora es incapaz de modificar la variable de la función (parámetro que se pasa).

El paso de argumentos por referencia se conoce como paso por variable. La función modifica el valor del parámetro pasado y devuelve el valor modificado a la función que la llama. La dirección de memoria del valor del parámetro se pasa a la función al compilar.

La lista de parámetros múltiples es utilizada para pasar a una función un número de parámetros indefinidos. El objeto que se utiliza para enviarlos es un objeto tipo lista (*List*). Esta lista permite enviar un número indeterminado de parámetros que en este caso serán objetos. Lenguajes orientados a objetos permiten utilizar este objeto. El objeto posee métodos y propiedades ya definidas. El parámetro que recibe la lista debe ser de tipo lista.

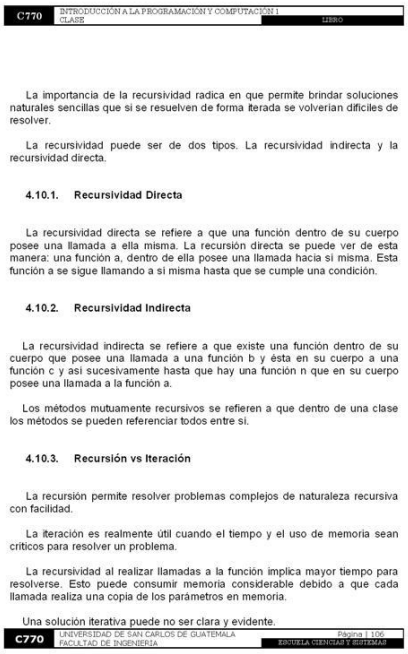
El problema con esta lista es que los valores en ella pueden ser modificados. Esto se evita si se utiliza la propiedad sin modificación de la lista. Esta propiedad permite que la lista no sea modificada a partir del momento en que se le indica sin modificar. Esto se realiza antes de enviar la lista como parámetro a la función ó procedimiento.

Los parámetros pueden ser declarados como tipo final. Esto permite incorporar seguridad debido a que se indica que el valor no puede ser alterado por la función. Esto quiere decir que ya sea por referencia o por valor, el parámetro final no cambiará. Esto debido a que será manejado como de sólo lectura.

4.10. Recursividad o Recursión

La recursividad es la propiedad que tiene una función de poder llamarse a si misma. Una función recursiva es una función que posee sentencias las cuales hacen llamadas a la propia función.

La recursividad surge de la necesidad de que las funciones se llamen a si mismas.



La importancia de la recursividad radica en que permite brindar soluciones naturales sencillas que si se resuelven de forma iterada se volverían difíciles de resolver.

La recursividad puede ser de dos tipos. La recursividad indirecta y la recursividad directa.

4.10.1. Recursividad Directa

La recursividad directa se refiere a que una función dentro de su cuerpo posee una llamada a ella misma. La recursión directa se puede ver de esta manera: una función a, dentro de ella posee una llamada hacia si misma. Esta función a se sigue llamando a si misma hasta que se cumple una condición.

4.10.2. Recursividad Indirecta

La recursividad indirecta se refiere a que existe una función dentro de su cuerpo que posee una llamada a una función b y ésta en su cuerpo a una función c y así sucesivamente hasta que hay una función n que en su cuerpo posee una llamada a la función a.

Los métodos mutuamente recursivos se refieren a que dentro de una clase los métodos se pueden referenciar todos entre si.

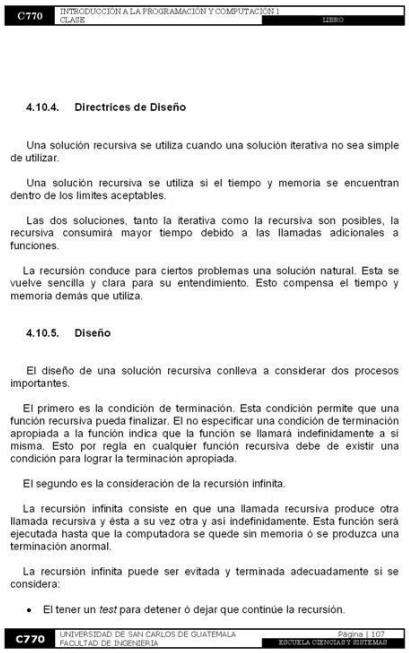
4.10.3. Recursión vs Iteración

La recursión permite resolver problemas complejos de naturaleza recursiva con facilidad.

La iteración es realmente útil cuando el tiempo y el uso de memoria sean críticos para resolver un problema.

La recursividad al realizar llamadas a la función implica mayor tiempo para resolverse. Esto puede consumir memoria considerable debido a que cada llamada realiza una copia de los parámetros en memoria.

Una solución iterativa puede no ser clara y evidente.



4.10.4. Directrices de Diseño

Una solución recursiva se utiliza cuando una solución iterativa no sea simple de utilizar.

Una solución recursiva se utiliza si el tiempo y memoria se encuentran dentro de los límites aceptables.

Las dos soluciones, tanto la iterativa como la recursiva son posibles, la recursiva consumirá mayor tiempo debido a las llamadas adicionales a funciones.

La recursión conduce para ciertos problemas una solución natural. Esta se vuelve sencilla y clara para su entendimiento. Esto compensa el tiempo y memoria demás que utiliza.

4.10.5. Diseño

El diseño de una solución recursiva conlleva a considerar dos procesos importantes.

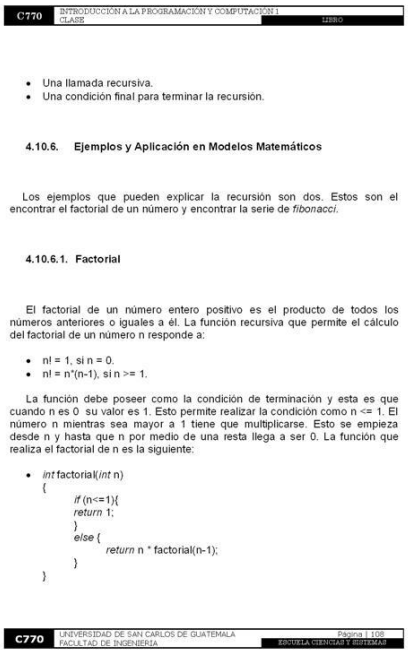
El primero es la condición de terminación. Esta condición permite que una función recursiva pueda finalizar. El no especificar una condición de terminación apropiada a la función indica que la función se llamará indefinidamente a si misma. Esto por regla en cualquier función recursiva debe de existir una condición para lograr la terminación apropiada.

El segundo es la consideración de la recursión infinita.

La recursión infinita consiste en que una llamada recursiva produce otra llamada recursiva y ésta a su vez otra y así indefinidamente. Esta función será ejecutada hasta que la computadora se quede sin memoria ó se produzca una terminación anormal.

La recursión infinita puede ser evitada y terminada adecuadamente si se considera:

- El tener un *test* para detener ó dejar que continúe la recursión.



- Una llamada recursiva.
- Una condición final para terminar la recursión.

4.10.6. Ejemplos y Aplicación en Modelos Matemáticos

Los ejemplos que pueden explicar la recursión son dos. Estos son el encontrar el factorial de un número y encontrar la serie de *fibonacci*.

4.10.6.1. Factorial

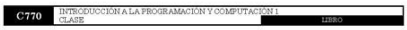
El factorial de un número entero positivo es el producto de todos los números anteriores o iguales a él. La función recursiva que permite el cálculo del factorial de un número n responde a:

- n! = 1, si n = 0.
- n! = n*(n-1), si n >= 1.

La función debe poseer como la condición de terminación y esta es que cuando n es 0 su valor es 1. Esto permite realizar la condición como n <= 1. El número n mientras sea mayor a 1 tiene que multiplicarse. Esto se empieza desde n y hasta que n por medio de una resta llega a ser 0. La función que realiza el factorial de n es la siguiente:

- ```
  int factorial(int n)
  {
      if (n<=1){
      return 1;
      }
      else {
          return n * factorial(n-1);
      }
  }
  ```
  ```

Figura 37. Páginas 109 – 112 Libro “Programación principiantes”



4.10.6.2. *Fibonacci*

La serie de *fibonacci* es una sucesión de números infinitos. Esta sucesión es la siguiente: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,… Esto quiere decir que:

- *Fibonacci*(1) = 1
- *Fibonacci*(2) = 1
- *Fibonacci*(3) = 2
- *Fibonacci*(4) = 3
- *Fibonacci*(5) = 5
- *Fibonacci*(6) = 8

La función que realiza la serie de *fibonacci* para un número n posee la condición de terminación donde n es igual a cero ó n es igual a 1. Esto hace que el valor de retorno sea n. El caso en que no se cumpla esta condición, se realizan dos llamadas a la misma función *fibonacci*. La primera como parámetro n-1 y la segunda con n-2. La función sería la siguiente:

- ```
  int Fibonacci(int n)
  {
      if (n == 0 || n == 1){
      return n;
      }
      else {
          return Fibonacci(n-1) + Fibonacci(n-2);
      }
  }
  ```


5. ESTRUCTURAS ALGORÍTMICAS

5.1. Arreglos de Datos

Los arreglos de datos se conocen como *arrays*. Un arreglo es una agrupación de valores del mismo tipo. Los elementos de un arreglo pueden ser cualquier tipo de objeto y solo puede estar formado por objetos del mismo tipo.

La longitud de un arreglo es determinada al momento en que se declara. Lenguajes orientados a objetos utilizan el método longitud para devolver el tamaño del arreglo.

Los elementos del arreglo se almacenarán en bloques contiguos de memoria. La indexación permitirá, dependiendo de cual se elija, apartar memoria desde la posición [0].

5.1.1. Indexación

Las posiciones que ocupan los elementos dentro de un arreglo se llaman índice y son correlativas.

Las formas básicas de indexar son tres. Estas son:

- Indexación base cero: esta permite que el primer elemento se encuentre en la posición [0] y la última posición corresponda a [n-1].
- Indexación base uno: esta permite que el primer elemento se encuentre en la posición [1] y la última posición corresponda a [n].
- Indexación base n: esta permite dejar libre la elección del primer elemento. Esto quiere decir que se puede colocar cualquier número para iniciar el arreglo.

5.1.2. Conceptos Generales

5.1.2.1. Declaración

Un arreglo se debe declarar antes de utilizarlo.

Un arreglo se declara colocando el tipo de datos que contendrá el arreglo, posteriormente el nombre seguido de corchetes.

Las formas de declaración pueden variar. Estas variaciones pueden ser:

- tipo nombreArray[] = new tipo[tamaño];
- tipo []nombreArray = new tipo[tamaño];
- tipo nombreArray2[] = {elemento$_1$, elemento$_2$, elemento$_3$,..., elemento$_n$};

5.1.2.2. Subíndices

El índice que identifica los elementos de un arreglo generalmente se denomina subíndice. Esto permite lo siguiente:

- elemento$_1$ = elemento[1]
- elemento$_5$ = elemento[5]

5.1.2.3. Atributo Longitud

Los lenguajes orientados a objetos poseen métodos para cada uno de los objetos que utilizan. Esto permite que los arreglos posean métodos asociados a ellos.

El método longitud (length), permite devolver la longitud como un valor entero de un arreglo.

La longitud de un arreglo no cambiará una vez el arreglo ha sido creado. La sintaxis de éste método es la siguiente:

- *int* longitud = nombreArreglo.*lenght*;

5.1.2.4. Inicialización

Un arreglo una vez declarado se le debe asignar valores. La asignación de valores depende de la indexación. Esta generalmente utiliza la indexación basada en cero.

La inicialización puede utilizar un bucle *for* para inicializar el arreglo a partir de 0 hasta n-1. La otra forma para inicializar el arreglo es asignar valores dentro de llaves al momento de su declaración.

5.1.2.5. Copia de Arreglos

Los arreglos pueden copiar su contenido completo en otros arreglos. La copia de los arreglos puede ser de dos formas. Las formas son una manual y una automática.

La forma manual implica recorrer el arreglo que se desea copiar desde el principio hasta que se alcance la última posición del arreglo. Esto quiere decir que cada posición será copiada cada una a una posición de otro arreglo. Las posiciones se recorren por medio de un bucle *for*.

Ejemplo 5.1.

Realizar la declaración de dos arreglos. El primer arreglo debe ser llenado de números aleatorios. Este arreglo posteriormente debe ser copiado completo al segundo arreglo. La cantidad de números que deben ser guardados en cada arreglo son 10.

El pseudocódigo que brinda la solución a este problema es el siguiente:

- ```
  //declaración de las variables y de los arreglos
  final int N = 10;
  int Arreglo1[] = new int[N];
  int Arreglo2[] = new int[N];
  //se realiza la inserción de números aleatorios
  for (int i = 0; i < N; i++)
  {
      Arreglo1[i] = Aleatorio()*99+1;
  }
  ```

Figura 38. Páginas 113 – 116 Libro “Programación principiantes”


```
//se realiza la copia de elementos desde Arreglo1 hacia Arreglo2
for (int i = 0; i < N; i++)
{
        Arreglo2[i] = Arreglo1[i];
}
```

La forma automática de copia de arreglos depende fundamentalmente del lenguaje de programación orientado a objetos que se esté utilizando. El lenguaje será quien indique el nombre del método y la cantidad de parámetros que se deben pasar.

El pseudocódigo de esta función sería:

- CopiarArreglo(Objeto Fuente, int PosFuente, Objeto Destino, int PosDestino, int Longitud)

Este método permite solicitar la copia de un objeto Fuente que corresponde al arreglo origen. PosFuente indica la posición inicial del arreglo origen. El objeto Destino corresponde al arreglo destino que recibirá los elementos del arreglo Fuente. PosDestino indica la posición final del arreglo destino. Longitud permite indicar el número de elementos que se han copiado.

5.1.3. Arreglos Bidimensionales

Los arreglos bidimensionales se conocen como matrices.

Los arreglos bidimensionales poseen dos dimensiones. Este tipo de arreglos se ve como una tabla de cálculo, donde existen filas y columnas.

Un arreglo bidimensional se declara con [][]. El primer par de corchetes se puede tomar como las filas y el segundo par como las columnas.

La declaración es entonces:

- tipo nombreArreglo[][] = new tipo[tamañofila][tamañocol];
- tipo [][] nombreArreglo = new tipo[tamañofila][tamañocol];

Los arreglos bidimensionales también se pueden declarar e inicializar con elementos:

- *int* arregloB[][] = {{1,2,3},{4,5,6}};

La representación en memoria de un arreglo bidimensional indica que las posiciones son consecutivas empezando por las filas y posteriormente las columnas. Esto quiere decir que al definirse un arreglo de tipo entero de 3 filas y 2 columnas, las posiciones relativas de memoria estarían representadas en la tabla 5.1. El tamaño ocupado por un tipo entero son 4 bytes.

Tabla 5.1. Representación en memoria arreglo bidimensional

Elemento	Posición Relativa en Memoria (bytes)
arregloB[0][0]	0
arregloB[0][1]	4
arregloB[1][0]	8
arregloB[1][1]	12
arregloB[2][0]	16
arregloB[2][1]	20

El atributo longitud en un arreglo bidimensional contiene el total de filas. El atributo longitud de cada fila contiene el total de columnas.

El acceso a cada elemento de un arreglo bidimensional se realiza por medio de:

- variable = nombreArreglo[fila][col];
- nombreArreglo[fila][col] = valor;

El acceso automático de todos los elementos en un arreglo bidimensional se basa en bucles *for* anidados. El pseudocódigo que responde a esto sería:

```
- for (fila=0; fila< arregloB.length; fila++){
        for(col=0; col<arregloB[fila].length; col++){
        //se hace algo con arregloB[fila][col];
        }
  }
```

5.1.4. Arreglos de n-Dimensiones

Los arreglos de n-dimensiones se conocen como multidimensionales. Un arreglo multidimensional utilizado comúnmente es el de tres dimensiones y se conoce como cubo.

Los arreglos de más de tres dimensiones raramente son utilizados.

Los arreglos tridimensionales son vistos como arreglos bidimensionales a los que se les añade profundidad. La declaración de estos arreglos es:

- tipo nombreArreglo[][][] = new int[filas][columnas][profundidad];

5.1.4.1. Aplicación Practica

Los cubos pueden utilizarse de forma que se apeguen a la realidad. La aplicación práctica para un cubo es la de un libro. Esto quiere decir que un libro puede ser representado como un arreglo tridimensional.

Un libro contiene páginas que pueden ser consideradas como un arreglo bidimensional constituido por filas y columnas. Las filas corresponden a un renglón de cada página. Las columnas corresponden a un carácter en la fila de cada página. El tipo de datos del arreglo corresponderá al tipo carácter. El libro por ejemplo posee 300 páginas con 25 renglones por página y 80 caracteres. Las páginas corresponden entonces a la tercera dimensión del arreglo pues estas dan la profundidad. El pseudocódigo correspondiente a la declaración de este problema y a la forma de acceso es la siguiente:

```
- //declaración del arreglo libro
  char libro[][][] = new [25][80][300];
  //acceso a la información del arreglo libro
  for (pagina=0; pagina< 300; pagina++){
        for (fila=0; fila< 25; fila++){
              for (col=0; col< 80; col++){
              //realizar algo con
              //libro[fila][col][pagina];
              }
        }
  }
```


5.1.5. Manejo de Arreglos como Parámetros

Los arreglos se pueden enviar como parámetros a un método colocando solamente el nombre del arreglo sin los corchetes. La función que recibe el parámetro de tipo arreglo debe de definir el arreglo en sus parámetros como:

- tipo NombreArreglo[]

El problema de enviar arreglos como parámetros es que éstos pueden ser pasados por medio de referencia. Esto quiere decir que si dentro del método se modifica un valor del arreglo éste será modificado en el arreglo que se pasa como parámetro. La recomendación es que si se utilizan como parámetros se debe tener cuidado de no modificar su referencia. El arreglo que se pasa como parámetro muchas veces necesita que se realicen operaciones sobre él. Estas operaciones no se deben reflejar en el arreglo. Esto se realiza creando otro arreglo dentro del método y se copia el contenido del arreglo que pasa como parámetro para que las operaciones sean realizados por el arreglo copia.

5.2. Cadenas de Caracteres

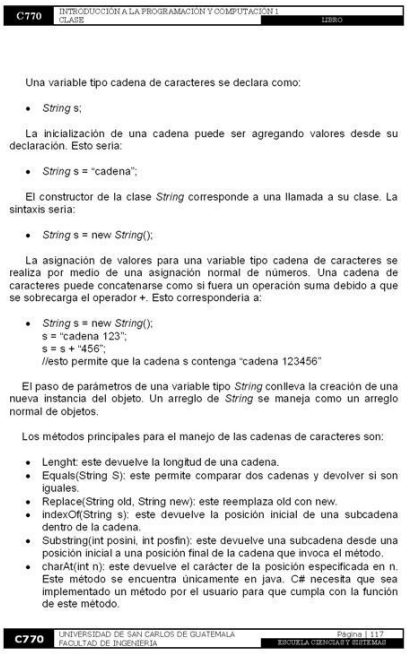
Una cadena de caracteres es un vector de caracteres. Los lenguajes orientados a objetos presentan las cadenas de caracteres como objetos que poseen sus propios atributos y métodos. Las cadenas de caracteres no se encuentran limitadas. Estas pueden ser indexadas por enteros de tal forma que se obtiene que el límite fijado sea de 2^{32} caracteres. Esto da como resultado una cadena de más de 4 Gb en memoria.

La diferencia entre un arreglo de caracteres y una cadena de caracteres (*String*) radica en que la clase cadena posee métodos especializados que permiten la fácil manipulación de los datos. Esto contrasta con los arreglos de caracteres.

5.2.1. Cadenas Estáticas

La clase *String* dependiendo del lenguaje pertenecerá a ciertas clases. Java utiliza la clase java.lang.*String* para manejar las cadenas de caracteres. C# utiliza la clase *System.String* para el manejo de las cadenas de caracteres.

Figura 39. Páginas 117 – 120 Libro “Programación principiantes”



Una variable tipo cadena de caracteres se declara como:

- *String* s;

La inicialización de una cadena puede ser agregando valores desde su declaración. Esto seria:

- *String* s = “cadena”;

El constructor de la clase *String* corresponde a una llamada a su clase. La sintaxis seria:

- *String* s = new *String*();

La asignación de valores para una variable tipo cadena de caracteres se realiza por medio de una asignación normal de números. Una cadena de caracteres puede concatenarse como si fuera un operación suma debido a que se sobrecarga el operador +. Esto corresponderia a:

- *String* s = new *String*();
  s = “cadena 123”;
  s = s + “456”;
  //esto permite que la cadena s contenga “cadena 123456”

El paso de parámetros de una variable tipo *String* conlleva la creación de una nueva instancia del objeto. Un arreglo de *String* se maneja como un arreglo normal de objetos.

Los métodos principales para el manejo de las cadenas de caracteres son:

- Lenght: este devuelve la longitud de una cadena.
- Equals(String S): este permite comparar dos cadenas y devolver si son iguales.
- Replace(String old, String new): este reemplaza old con new.
- indexOf(String s): este devuelve la posición inicial de una subcadena dentro de la cadena.
- Substring(int posini, int posfin): este devuelve una subcadena desde una posición inicial a una posición final de la cadena que invoca el método.
- charAt(int n): este devuelve el carácter de la posición especificada en n. Este método se encuentra únicamente en java. C# necesita que sea implementado un método por el usuario para que cumpla con la función de este método.

5.2.2. Cadenas Dinámicas

Las cadenas dinámicas permiten representar una cadena cuyo tamaño puede ser variable. Esto quiere decir que almacenará una gran cantidad de texto. Esta cantidad hace que no sea apropiado utilizar una cadena dinámica. Esto es debido a que cada vez que se utiliza un método en la clase *String* se crea un nuevo objeto *String* en memoria. Esto se evita utilizando cadenas dinámicas ya que ellas no necesitan crear un nuevo objeto en memoria.

Las cadenas dinámicas mejoran el rendimiento considerablemente. Estas cadenas también aunque consumen más recursos que una cadena estática se recomiendan para el manejo grande de información.

Los lenguajes orientados a objetos poseen diversos nombres para sus clases de cadenas dinámicas.

Java conoce a las cadenas dinámicas como *StringBuffer*. C# conoce a las cadenas dinámicas como *StringBuilder*.

5.2.2.1. Declaración

La declaración de un objeto de tipo cadena dinámica dependerá del lenguaje.

La declaración en Java seria:

- StringBuffer nombreSD = new StringBuffer();

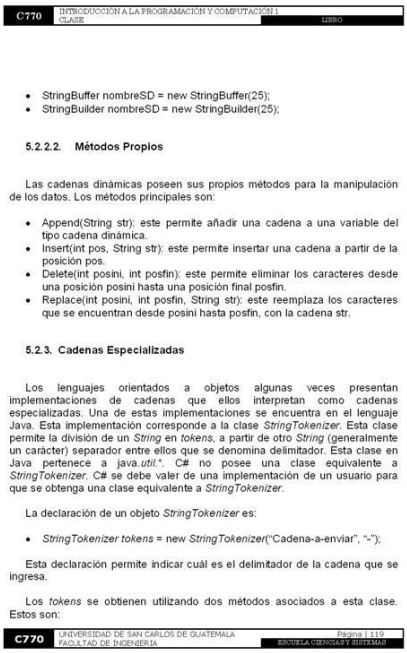
La declaración en C# seria:

- StringBuilder nombreSD = new StringBuilder();

Las cadenas dinámicas también pueden ser creadas a partir de una cadena estática, esto corresponderia a:

- StringBuffer nombreSD = new StringBuffer(“Hola”);
- StringBuilder nombreSD = new StringBuilder(“Hola”);

Las cadenas dinámicas también pueden ser declaradas especificando su tamaño.



- StringBuffer nombreSD = new StringBuffer(25);
- StringBuilder nombreSD = new StringBuilder(25);

5.2.2.2. Métodos Propios

Las cadenas dinámicas poseen sus propios métodos para la manipulación de los datos. Los métodos principales son:

- Append(String str): este permite añadir una cadena a una variable del tipo cadena dinámica.
- Insert(int pos, String str): este permite insertar una cadena a partir de la posición pos.
- Delete(int posini, int posfin): este permite eliminar los caracteres desde una posición posini hasta una posición final posfin.
- Replace(int posini, int posfin, String str): este reemplaza los caracteres que se encuentran desde posini hasta posfin, con la cadena str.

5.2.3. Cadenas Especializadas

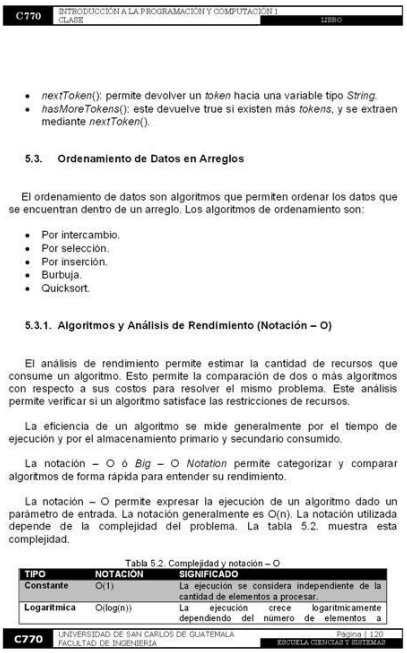
Los lenguajes orientados a objetos algunas veces presentan implementaciones de cadenas que ellos interpretan como cadenas especializadas. Una de estas implementaciones se encuentra en el lenguaje Java. Esta implementación corresponde a la clase *StringTokenizer*. Esta clase permite la división de un *String* en *tokens*, a partir de otro *String* (generalmente un carácter) separador entre ellos que se denomina delimitador. Esta clase en Java pertenece a *java.util.**. C# no posee una clase equivalente a *StringTokenizer*. C# se debe valer de una implementación de un usuario para que se obtenga una clase equivalente a *StringTokenizer*.

La declaración de un objeto *StringTokenizer* es:

- *StringTokenizer tokens* = new *StringTokenizer*(“Cadena-a-enviar”, “-”);

Esta declaración permite indicar cuál es el delimitador de la cadena que se ingresa.

Los *tokens* se obtienen utilizando dos métodos asociados a esta clase. Estos son:



- *nextToken*(): permite devolver un *token* hacia una variable tipo *String*.
- *hasMoreTokens*(): este devuelve true si existen más *tokens*, y se extraen mediante *nextToken*().

5.3. Ordenamiento de Datos en Arreglos

El ordenamiento de datos son algoritmos que permiten ordenar los datos que se encuentran dentro de un arreglo. Los algoritmos de ordenamiento son:

- Por intercambio.
- Por selección.
- Por inserción.
- Burbuja.
- Quicksort.

5.3.1. Algoritmos y Análisis de Rendimiento (Notación – O)

El análisis de rendimiento permite estimar la cantidad de recursos que consume un algoritmo. Esto permite la comparación de dos o más algoritmos con respecto a sus costos para resolver el mismo problema. Este análisis permite verificar si un algoritmo satisface las restricciones de recursos.

La eficiencia de un algoritmo se mide generalmente por el tiempo de ejecución y por el almacenamiento primario y secundario consumido.

La notación – O ó *Big* – O *Notation* permite categorizar y comparar algoritmos de forma rápida para entender su rendimiento.

La notación – O permite expresar la ejecución de un algoritmo dado un parámetro de entrada. La notación generalmente es O(n). La notación utilizada depende de la complejidad del problema. La tabla 5.2. muestra esta complejidad.

Tabla 5.2. Complejidad y notación – O

TIPO	NOTACION	SIGNIFICADO
Constante	O(1)	La ejecución se considera independiente de la cantidad de elementos a procesar.
Logaritmica	O(log(n))	La ejecución crece logaritmicamente dependiendo del número de elementos a

Figura 40. Páginas 121 – 124 Libro “Programación principiantes”

		procesar.
Lineal	O(n)	La ejecución crece linealmente dependiendo del número de elementos a procesar.
N log N	O(n*log(n))	La ejecución crece como un producto dependiendo de la complejidad lineal y logaritmica.
Cuadrática	O(n^2)	La ejecución crece cuadráticamente dependiendo del número de elementos a procesar.

5.3.2. Ordenamiento por Intercambio

El algoritmo por ordenamiento por intercambio se conoce también como SWAP. Este algoritmo es el más sencillo en su implementación.

El algoritmo se basa en ordenar los elementos de una lista de forma ascendente. La base del algoritmo es la comparación del elemento inferior de la lista con los restantes y efectuando el intercambio de posición cuando el orden que resulta de la comparación no es el correcto.

El algoritmo se realizará n-1 pasadas para ordenar la lista, donde n es el total de elementos den la lista.

El algoritmo que responde a este método de ordenamiento es el siguiente:

```
void OrdenarSWAP(double lista[])
{
    int N = lista.lenght;
    double valaux = 0;
    for (int i=0; i<N-1; i++)
    {
        for(int j=i+1; j<N; j++)
        {
            if (lista[i]> lista[j])
            {
                valaux = lista[i];
                lista[i] = lista[j];
                lista[j] = valaux;
            }
        }
    }
```



```
}
```

5.3.3. Ordenamiento por Selección

El algoritmo de selección se fundamenta en tres pasos. Estos pasos permiten implementar el algoritmo para que ordene los datos. Los pasos son:

- Se selecciona el elemento más pequeño de la lista (L). Ahora se intercambia por L[0], de esta forma el valor más pequeño se encuentra en la primera posición de la lista.
- Ahora se deben considerar L[1], L[2], L[3], …. De estas se selecciona el elemento más pequeño y se realiza el intercambio con L[1].
- Se prosigue el proceso anulando las posiciones donde ya se ha intercambiado el menor elemento. En este caso la posición 0 y la 1 ya han sido anuladas se prosigue encontrando el mínimo e intercambiándolo en las restantes hasta que todas las posiciones sean anuladas.

El algoritmo que realiza el ordenamiento por selección es:

```
void ordenarSeleccion (double lista[])
{
    double tmp;
    int posmin;
    int N = lista.length;
    for (int i=0; i<N-1; i++)
    {
        posmin = i;
        for (int j=i+1; j<N; j++)
        {
            if (lista[j]<lista[posmin])
            {
                posmin=j;
            }
        }
        tmp = lista[i];
        lista[i] = lista[posmin];
        lista[posmin] = tmp;
    }
}
```


5.3.4. Ordenamiento por Inserción

El algoritmo de ordenamiento por inserción al igual que el de selección se fundamenta en una serie de pasos. Estos pasos son:

- El elemento en lista[0] está considerado ordenado, es decir que la lista de inicio posee un elemento.
- Se inserta lista[1] en la posición correcta; ya sea delante o atrás de lista[0]. La lista es explorada desde lista[i] hasta lista[n] buscando la posición correcta para insertar dentro de una lista ordenada.
- Por cada iteración se debe mover hacia debajo de la lista una posición todos los elementos que sean mayores a la posición a insertar para dejar vacía esa posición.
- Se debe insertar el elemento en la posición correcta.

El algoritmo que se utiliza para representar el ordenamiento por inserción es:

```
void ordenarInsercion (double lista[])
{
    double tmp;
    int j;
    int N = lista.length;
    for (int i=1; i<N; i++)
    {
        tmp = lista[i];

        for (j=i; (j>0)&&(tmp<lista[j-1]);j--)
        {
            lista[j] = lista[j-1];
        }
        lista [j] = tmp;
    }
}
```

5.3.5. Ordenamiento de Burbuja

El ordenamiento de burbuja también se conoce como *bubblesort*. El algoritmo de burbuja es el más sencillo y fácil de comprender. Este algoritmo es el menos eficiente.

El ordenamiento de burbuja se denomina también ordenación por hundimiento debido a que los valores más pequeños van subiendo en la lista gradualmente y los valores mayores van al fondo de la lista.

El algoritmo que se utiliza para el ordenamiento de burbuja es el siguiente:

```
void ordenarBurbuja (double lista[])
{
    double tmp;
    int N = lista.length;
    for (int i=1; i<N; i++)
    {
        for (int j=N-1; j>=i; j--)
        {
            if (lista[j]<lista[j-1])
            {
                tmp = lista[j];
                lista[j] = lista[j-1];
                lista[j-1] = tmp;
            }
        }
    }
}
```

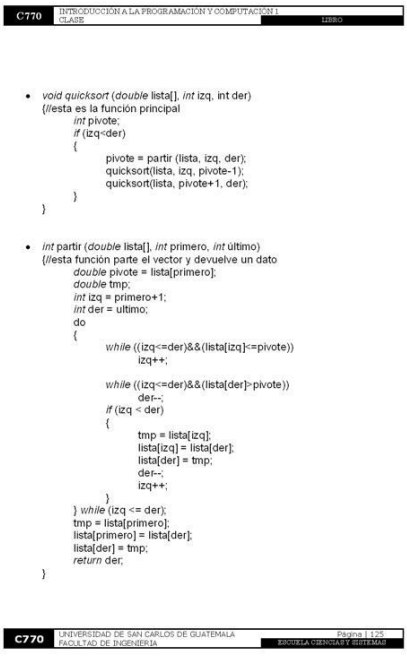
5.3.6. Ordenamiento *QuickSort*

El algoritmo de ordenamiento rápido se basa en una función recursiva. El algoritmo de ordenamiento rápido se basa en los siguientes pasos:

- Se debe tomar un elemento de la lista. Este será denominado pivote.
- El vector será dividido de forma que los elementos a la izquierda del pivote sean menores que él, los de la derecha serán mayores a él.
- Las dos partes serán ordenadas por separado.
- La función principal es la de *quicksort* y esta utiliza una función que parta este vector.

El algoritmo correspondiente a los pasos es:

Figura 41. Páginas 125 – 128 Libro “Programación principiantes”



- *void quicksort* (*double* lista[], *int* izq, int der)
{//esta es la función principal
int pivote;
if (izq<der)
{
pivote = partir (lista, izq, der);
quicksort(lista, izq, pivote-1);
quicksort(lista, pivote+1, der);
}
}

- *int* partir (*double* lista[], *int* primero, *int* último)
{//esta función parte el vector y devuelve un dato
double pivote = lista[primero];
double tmp;
int izq = primero+1;
int der = ultimo;
do
{
while ((izq<=der)&&(lista[izq]<=pivote))
izq++;

while ((izq<=der)&&(lista[der]>pivote))
der--;
if (izq < der)
{
tmp = lista[izq];
lista[izq] = lista[der];
lista[der] = tmp;
der--;
izq++;
}
} *while* (izq <= der);
tmp = lista[primero];
lista[primero] = lista[der];
lista[der] = tmp;
return der;
}

5.4. Búsqueda de Datos en Arreglos

5.4.1. Búsqueda Secuencial

Una búsqueda secuencial no es más que buscar un elemento de una lista utilizando un valor de una lista utilizando un valor de destino denominado clave. Esta búsqueda se denomina lineal. La búsqueda lineal recorre la lista uno por uno encontrando el elemento deseado.

El análisis de rendimiento de este algoritmo conlleva considerar el mejor y el peor caso. El mejor caso ocurre cuando se da la coincidencia que el elemento buscado se encuentra en la primera posición de la lista. El peor de los casos es cuando el elemento buscado se encuentra al final de la lista.

El algoritmo que se utiliza para realizar una búsqueda secuencial es:

- *int* buscarSecuencial(*double* vector[], *double* dato)
{
int N = lista.*length*;
int pos = -1;
for (*int* i=0; (i<N) && (pos==-1); i++)
{
if (lista[i]==dato)
{
pos=i;
}
}
return pos;
}

5.4.2. Búsqueda Binaria

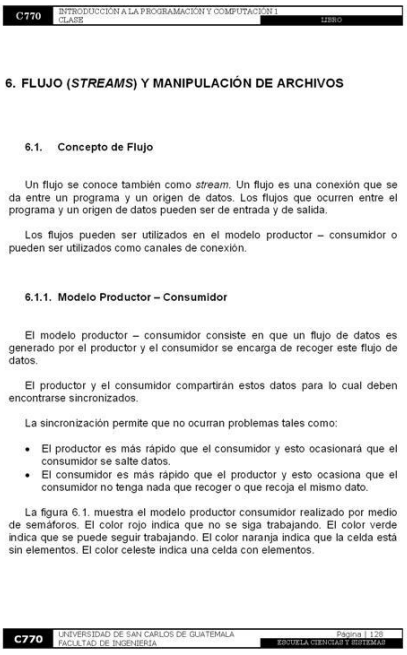
La búsqueda binaria se caracteriza debido a que necesita que la lista esté ordenada. Una búsqueda de este tipo compara el elemento de la mitad de la lista con el dato a buscar. Esto ocasiona que si ambos datos son iguales se encontró el dato, si el dato es mayor se busca el dato en esa mitad de la lista.

La búsqueda binaria se basa en que si el dato es menor que el elemento del centro de la lista, se busca en la primera mitad de la lista. Esto es realizado hasta encontrar el dato deseado.

El mejor de los casos ocurre cuando se encuentra el elemento en el punto central de la lista. El peor de los casos se harán 1+log2n iteraciones. El peor de los casos corresponde a O(log2n). El mejor de los casos es O(1).

El algoritmo que realiza una búsqueda secuencial es el siguiente:

- *int* buscarBinario(*double* lista[], *int* izq, *int* der, *double* buscado)
{
int centro = (izq+der)/2;
if (izq>der)
{
return -1;
}
else
{
if (buscado == lista[centro])
{
return centro;
}
else
{
if (buscado < lista[centro])
{
return buscarBinario(lista, izq, centro-1, buscado);
}
else
{
return buscarBinario(lista, centro+1, der, buscado);
}
}
}
}



6. FLUJO (*STREAMS*) Y MANIPULACIÓN DE ARCHIVOS

6.1. Concepto de Flujo

Un flujo se conoce también como *stream*. Un flujo es una conexión que se da entre un programa y un origen de datos. Los flujos que ocurren entre el programa y un origen de datos pueden ser de entrada y de salida.

Los flujos pueden ser utilizados en el modelo productor – consumidor o pueden ser utilizados como canales de conexión.

6.1.1. Modelo Productor – Consumidor

El modelo productor – consumidor consiste en que un flujo de datos es generado por el productor y el consumidor se encarga de recoger este flujo de datos.

El productor y el consumidor compartirán estos datos para lo cual deben encontrarse sincronizados.

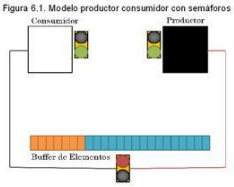
La sincronización permite que no ocurran problemas tales como:

- El productor es más rápido que el consumidor y esto ocasionará que el consumidor se salte datos.
- El consumidor es más rápido que el productor y esto ocasiona que el consumidor no tenga nada que recoger o que recoja el mismo dato.

La figura 6.1. muestra el modelo productor consumidor realizado por medio de semáforos. El color rojo indica que no se siga trabajando. El color verde indica que se puede seguir trabajando. El color naranja indica que la celda está sin elementos. El color celeste indica una celda con elementos.

Figura 42. Páginas 129 – 132 Libro “Programación principiantes”

Figura 6.1. Modelo productor consumidor con semáforos



6.1.2. Canales de Conexión

Los canales de conexión se conocen como *pipes*. Los *pipes* son utilizados para dirigir la salida de un programa hacia la entrada de otro programa.

Los canales de conexión son útiles al momento de utilizar métodos que producen una salida que necesita ser utilizada como entrada en otro método.

Los canales de conexión permiten no utilizar archivos temporales que permitirán la comunicación entre los métodos implicados en la conexión.

Java utiliza las clases *PipedInputStream* y *PipedOutputStream* para manejar los canales de conexión. C# utiliza la clase *NamedPipeNative* para realizar el manejo de pipes.

6.2. Tipos de Flujos

Java maneja la jerarquía de los flujos de datos mediante clases. Estas dependen del tipo de datos:

- *char Unicode* de 16 *bits*.
  - *Reader*.
  - *Writer*.
- *Byte* de 8 *bit*.

  - *InputStream*.
  - *OutputStream*.

Las clases de manejo de E/S de flujo de datos se encuentran en el paquete java.io.

La clase *InputStream* se divide en:

- *FileInputStream*
- *PipedInputStream*
- *FilterInputStream*
- *LineNumberInputStream*
- *DataInputStream*
- *BufferedInputStream*
- *PushbackInputStream*
- *ByteArrayInputStream*
- *SequenceInputStream*
- *StringBufferInputStream*
- *ObjectInputStream*

La clase *OutputStream* se divide en:

- *FileOutputStream*
- *PipedOutputStream*
- *FilterOutputStream*
  - *DataOutputStream*
  - *BufferedOutputStream*
  - *PushbackOutputStream*
- *ByteArrayOutputStream*
- *ObjectOutputStream*

La jerarquía de clases de flujos de datos en C# se maneja dentro de la clase *System*.IO. Esta jerarquía comprende a las clases *MarshalByRefObject*, *BinaryReader*, *BinaryWriter*, estas comprenden:

- *BinaryReader Class*
- *BinaryWriter Class*
- *BufferedStream Class*
- *Directory Class*
- *DriveInfo Class*
- *File Class*
- *FileStream Class*

- *MemoryStream Class*
- *Path Class*
- *PipeException Class*
- *Stream Class*
- *StreamReader Class*
- *StreamWriter Class*
- *StringReader Class*
- *StringWriter Class*
- *TextReader Class*
- *TextWriter Class*
- *UnmanagedMemoryStream Class*

6.2.1. Definición y Operaciones para Archivos

Los archivos poseen una serie de operaciones que permiten su definición y su manipulación. Las clases de archivos permiten leer y escribir archivos. Los archivos permiten representar archivos de texto y archivos binarios. Los archivos de texto son accedidos secuencialmente (byte por byte). Los objetos tipo archivo permiten elegir el acceso a un byte, a varios o al archivo completo.

La sintaxis (el nombre de la clase archivo) variará dependiendo del lenguaje que se haya elegido. El caso de Java utiliza las clases *FileInputStream* y *FileOutputStream* para leer o escribir a otro flujo. C# utiliza la clase *BufferedStream* para leer o escribir a otro flujo.

Las operaciones que se realizan sobre la clase archivo son: declarar el archivo, abrir el archivo, leer el archivo, escribir el archivo y cerrar el archivo.

La declaración se realiza como una variable. La sintaxis es:

- ArchivoTexto miArchivoTxt;

La operación para abrir este archivo puede realizarse de dos formas:

- miArchivoTxt = new ArchivoTexto("c:/archivo.txt");
- La segunda correspondería a crear un objeto Archivo y éste es el que se abre:
  - Archivo miArchivo;
    miArchivo = new Archivo("c:/archivo.txt");
    miArchivoTxt = new ArchivoTexto(miArchivo);

La operación para cerrar el archivo se realiza por el método cerrar(). La sintaxis es:

- miArchivoTxt.cerrar();

La operación leer es una sola función que sobrecarga los atributos en ella. Esto significa que su sintaxis puede variar dependiendo el tipo de atributos que contenga. La sintaxis general lee un *byte* y devuelve -1 al final del flujo:

- *byte* miArchivo.leer();

La operación escribir es una sola función que sobrecarga los atributos en ella. Esto significa que su sintaxis puede variar dependiendo el tipo de atributos que contenga. La sintaxis general escribe un *byte*:

- miArchivo.write(*int* a);

6.2.2. Clases Filtro

Las clases filtro son utilizadas para manipular lecturas de datos a partir de un flujo.

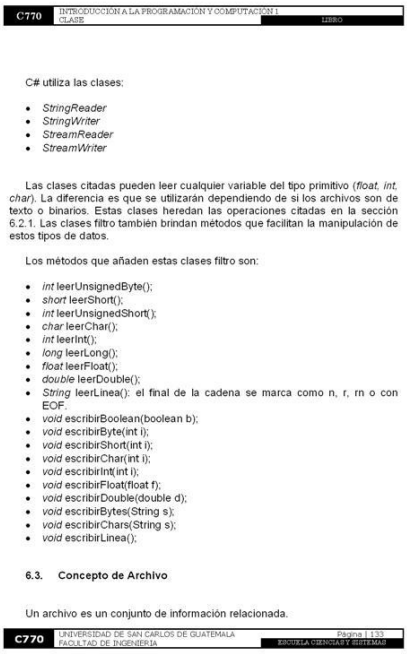
Las clases filtro permiten al usuario hacer una cadena utilizando múltiples flujos de entrada. Las operaciones que se le realizarán a la cadena permite crear una combinación de efecto sobre muchos filtros.

La utilización de estos flujos no necesita la conversión de los datos de byte a char mientras se escribe a un archivo.

Java maneja las clases filtro en dos clases. Estas son:

- *FilterInputStream*
  - *LineNumberInputStream*
  - *DataInputStream*
  - *BufferedInputStream*
  - *PushbackInputStream*
- *FilterOutputStream*
  - *DataOutputStream*
  - *BufferedOutputStream*
  - *PushbackOutputStream*

Figura 43. Páginas 133 – 136 Libro “Programación principiantes”



C# utiliza las clases:

- *StringReader*
- *StringWriter*
- *StreamReader*
- *StreamWriter*

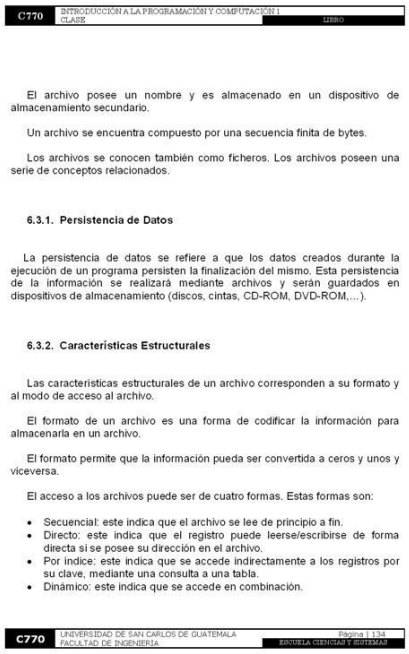
Las clases citadas pueden leer cualquier variable del tipo primitivo (*float, int, char*). La diferencia es que se utilizarán dependiendo de si los archivos son de texto o binarios. Estas clases heredan las operaciones citadas en la sección 6.2.1. Las clases filtro también brindan métodos que facilitan la manipulación de estos tipos de datos.

Los métodos que añaden estas clases filtro son:

- *int* leerUnsignedByte();
- *short* leerShort();
- *int* leerUnsignedShort();
- *char* leerChar();
- *int* leerInt();
- *long* leerLong();
- *float* leerFloat();
- *double* leerDouble();
- *String* leerLinea(): el final de la cadena se marca como n, r, rn o con EOF.
- *void* escribirBoolean(boolean b);
- *void* escribirByte(int i);
- *void* escribirShort(int i);
- *void* escribirChar(int i);
- *void* escribirInt(int i);
- *void* escribirFloat(float f);
- *void* escribirDouble(double d);
- *void* escribirBytes(String s);
- *void* escribirChars(String s);
- *void* escribirLinea();

6.3. Concepto de Archivo

Un archivo es un conjunto de información relacionada.



El archivo posee un nombre y es almacenado en un dispositivo de almacenamiento secundario.

Un archivo se encuentra compuesto por una secuencia finita de bytes.

Los archivos se conocen también como ficheros. Los archivos poseen una serie de conceptos relacionados.

6.3.1. Persistencia de Datos

La persistencia de datos se refiere a que los datos creados durante la ejecución de un programa persisten la finalización del mismo. Esta persistencia de la información se realizará mediante archivos y serán guardados en dispositivos de almacenamiento (discos, cintas, CD-ROM, DVD-ROM,…).

6.3.2. Características Estructurales

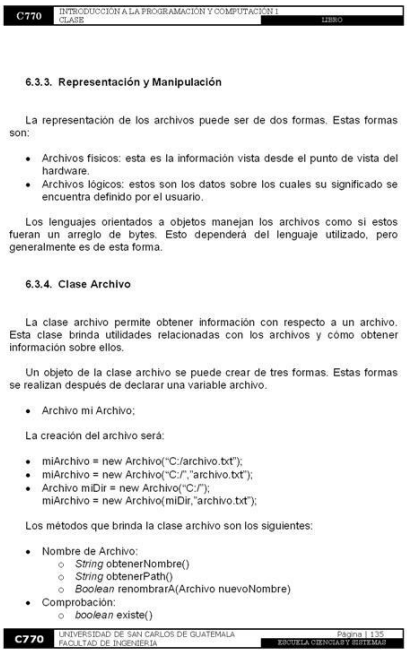
Las características estructurales de un archivo corresponden a su formato y al modo de acceso al archivo.

El formato de un archivo es una forma de codificar la información para almacenarla en un archivo.

El formato permite que la información pueda ser convertida a ceros y unos y viceversa.

El acceso a los archivos puede ser de cuatro formas. Estas formas son:

- Secuencial: este indica que el archivo se lee de principio a fin.
- Directo: este indica que el registro puede leerse/escribirse de forma directa si se posee su dirección en el archivo.
- Por índice: este indica que se accede indirectamente a los registros por su clave, mediante una consulta a una tabla.
- Dinámico: este indica que se accede en combinación.



6.3.3. Representación y Manipulación

La representación de los archivos puede ser de dos formas. Estas formas son:

- Archivos físicos: esta es la información vista desde el punto de vista del hardware.
- Archivos lógicos: estos son los datos sobre los cuales su significado se encuentra definido por el usuario.

Los lenguajes orientados a objetos manejan los archivos como si estos fueran un arreglo de bytes. Esto dependerá del lenguaje utilizado, pero generalmente es de esta forma.

6.3.4. Clase Archivo

La clase archivo permite obtener información con respecto a un archivo. Esta clase brinda utilidades relacionadas con los archivos y cómo obtener información sobre ellos.

Un objeto de la clase archivo se puede crear de tres formas. Estas formas se realizan después de declarar una variable archivo.

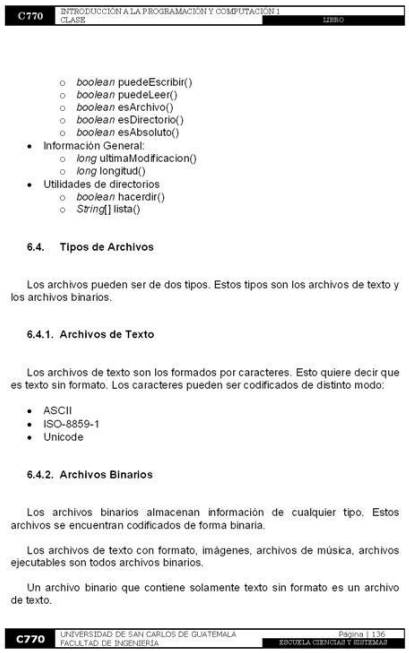
- Archivo mi Archivo;

La creación del archivo será:

- miArchivo = new Archivo("C:/archivo.txt");
- miArchivo = new Archivo("C://","archivo.txt");
- Archivo miDir = new Archivo("C:/");
  miArchivo = new Archivo(miDir,"archivo.txt");

Los métodos que brinda la clase archivo son los siguientes:

- Nombre de Archivo:
  - *String* obtenerNombre()
  - *String* obtenerPath()
  - *Boolean* renombrarA(Archivo nuevoNombre)
- Comprobación:
  - *boolean* existe()



  - *boolean* puedeEscribir()
  - *boolean* puedeLeer()
  - *boolean* esArchivo()
  - *boolean* esDirectorio()
  - *boolean* esAbsoluto()
- Información General:
  - *long* ultimaModificacion()
  - *long* longitud()
- Utilidades de directorios
  - *boolean* hacerdir()
  - *String*[] lista()

6.4. Tipos de Archivos

Los archivos pueden ser de dos tipos. Estos tipos son los archivos de texto y los archivos binarios.

6.4.1. Archivos de Texto

Los archivos de texto son los formados por caracteres. Esto quiere decir que es texto sin formato. Los caracteres pueden ser codificados de distinto modo:

- ASCII
- ISO-8859-1
- Unicode

6.4.2. Archivos Binarios

Los archivos binarios almacenan información de cualquier tipo. Estos archivos se encuentran codificados de forma binaria.

Los archivos de texto con formato, imágenes, archivos de música, archivos ejecutables son todos archivos binarios.

Un archivo binario que contiene solamente texto sin formato es un archivo de texto.

Figura 44. Páginas 137 – 140 Libro “Programación principiantes”

6.5. Estructura de Archivos

Los archivos se encuentran compuestos por un conjunto de registros. Los registros se consideran como una colección lógica de datos.

Un registro es un conjunto de datos que se encuentran organizados lógicamente. Los registros representan la unidad mínima de información de un archivo y de longitud fija.

Los registros se encuentran formados por campos. Un campo es la unidad de información (lógicamente significativa) más pequeña dentro de un archivo. Esto permite decir que un registro es un conjunto de campos agrupados.

6.5.1. Manipulación de Registros

La integridad de los campos dentro de los registros se puede mantener utilizando tres métodos. Estos métodos son:

- Obligar a que todos los campos posean una longitud predecible.
- Cada campo debe iniciar con un indicador de longitud.
- Usar un delimitador al final de cada campo.

6.5.2. Tamaño de Registros

El tamaño de los registros se pude determinar optando por las siguientes consideraciones:

- Se obliga a que todos los registros sean de longitud predecible. Esta longitud se mide en términos de bytes o del número de campos.
- Cada registro debe comenzar con un indicador de longitud que señale el tamaño en bytes del registro.
- Emplear un segundo archivo que contenga la dirección del byte de inicio de cada registro.
- Utilizar un delimitador al final de cada registro, para separarlo del siguiente.

6.6. Operaciones Básicas con Archivos

Las operaciones básicas que se pueden realizar con los archivos corresponden a abrir, cerrar, lectura, escritura, posicionamiento y localización al final del archivo.

Las operaciones de abrir, cerrar, lectura y escritura se detallan en la sección 6.2.1.

El abrir un archivo consiste en definir si éste será de lectura o escritura y si será del tipo texto o binario. Esto permite que posteriormente sea declarado e inicializado conforme a la sección 6.2.1. Una vez terminado de utilizarse el archivo debe ser cerrado. Esto se explica en la sección 6.2.1.

La lectura y escritura de un archivo dependerá del tipo de archivo que se haya creado. Las operaciones se detallan en la sección 6.2.1.

El posicionamiento dentro de un archivo generalmente se realiza de forma automática. Esto quiere decir que cada vez que se llama a un método de lectura o escritura el curso del archivo se mueve a la siguiente posición. Esto quiere decir que para llegar al final del archivo se deben leer todos los registros.

Los archivos de tipo aleatorios permiten que se pueda desplazarse dentro del archivo por medio de métodos especializados, lo que permite que no sea necesario recorrer todo el archivo.

El final de un archivo se encuentra determinado por EOF (*end of file*). Los métodos de lectura secuencial de archivos devuelven nulo al momento de alcanzar el final del archivo.

6.7. Clasificación Fundamental de Archivos según su Organización

La organización de un archivo puede ser:

- Secuencial
- Directa o Aleatoria

6.7.1. Organización Secuencial

La organización de un archivo puede estar constituida por registros que son guardados consecutivamente uno tras otro. Esta organización es conocida como organización secuencial.

La secuencia de acceso se encuentra determinada por su ordenamiento. Los registros en este tipo de organización deben ser leídos consecutivamente si se utiliza como entrada.

La lectura y escritura de los archivos secuenciales se encuentra dada por su ordenamiento secuencial. El leer un registro específico implica recorrer todo el archivo secuencialmente hasta encontrar el registro deseado. La escritura de estos archivos es similar. La clase archivo y la de flujos son archivos de acceso secuencial.

6.7.2. Organización Directa o Aleatoria

La organización directa se da cuando al momento de acceder a un registro “N” no tiene que pasar por los “N-1” registros anteriores.

La organización directa se basa en utilizar la llave del registro para localizarlo en el archivo.

La manipulación de un archivo de acceso directo implica que el archivo posea registros de tamaño fijo. Esto permite que se pueda añadir el apuntador interno del registro hacia cualquier registro determinado.

Los archivos de acceso aleatorio son los utilizados para accederse de forma directa o indirecta.

6.7.2.1. Clase Archivo Acceso Aleatorio

Un objeto de archivo de acceso aleatorio permite acceder a un archivo directamente, es decir saltando un registro a otro. Esto donde cada uno se encuentra en partes diferentes del archivo. Los archivos de acceso aleatorio son de tipo entrada/salida.

La declaración e inicialización de un archivo de acceso aleatorio es similar a la de un archivo secuencial. La diferencia radica en que la clase se llama ArchivoAleatorio y que esta además posee un parámetro que indica si es de lectura escritura o ambos.

- ArchivoAleatorio(*String* path, *String* Acceso)

El acceso puede ser:

- r: lectura.
- e: escritura.
- le: lectura y escritura.

Los objetos del tipo ArchivoAleatorio pueden utilizar los métodos de los objetos de archivos secuenciales. Esto significa que los métodos de lectura y escritura están disponibles para esta clase.

El posicionamiento dentro del archivo se vale de los siguientes métodos:

- *long* obtenerArchivoPuntero(): obtiene la posición del puntero en el archivo.
- *void* buscar(*long* pos): coloca el puntero del archivo en la posición pos. La posición se encuentra dada por un desplazamiento en bytes desde el comienzo del archivo. La posición 0 es el comienzo del archivo.
- *long* longitud(): devuelve la longitud del archivo. Esta posición marca el final del archivo.

Figura 45. Páginas 141 – 144 Libro “Programación principiantes”

7. TIPO DE DATOS ABSTRACTOS

7.1. Fundamentos Teóricos

7.1.1. Memoria Dinámica y Apuntadores

La memoria dinámica es un espacio de almacenamiento solicitado en tiempo de ejecución. Un apuntador es conocido también como puntero y es una variable que almacena una dirección de memoria.

Un puntero en C/C++ almacena la dirección de memoria. C/C++ deja utilizar punteros de tal forma que se puede corromper el sistema.

La protección de la memoria se deja al sistema operativo, el cual lanzaria un error de sistema al momento de que un puntero acceda a una posición no válida de memoria.

Las variables de referencia de Java almacenan la dirección de un objeto y este objeto se localiza en una dirección de memoria.

Java no permite corromper el sistema debido a que se lanzaria una excepción ya sea en tiempo de compilación o ejecución. Java al utilizar las referencias permite la protección del sistema y no deja que el sistema operativo devuelva el error.

7.1.2. Concepto de *NULL*

NULL también es considerada como *NILL*. Estos significan nulo. *NULL* es una constante reservada que representa una referencia *void*. Esto se observa en punteros que no apuntan a nada.

Un *NULL* internamente es un 0 binario, pero en lenguaje de alto nivel es distinto de 0 conteniendo cualquier representación.

Los objetos NULL al ser ejecutados despliegan una excepción. Esta excepción variará dependiendo del lenguaje. Java la utiliza *NullPointerException*. C# utiliza *ArgumentNullException*. Esto corresponde a

lenguajes orientados a objetos. La referencia NULL es distinta a una referencia a un objeto vacío.

Un objeto que no sea nulo puede realizar sus métodos. La sintaxis de esto es:

- ```
  if (objX != null)
  {
      objX.hacerAlgo();
  }
  ```

La sintaxis anterior se puede convertir en:

- ```
  objX!.hacerAlgo();
  ```

7.1.3. Introducción a la Clasificación de Estructuras

Las estructuras dinámicas de datos son aquellas que permiten un conjunto de elementos relacionados donde no importa la cantidad de elementos. Esto significa que el tamaño de estas estructuras no se define al principio debido a que crecen conforme se añaden elementos.

Los elementos en estas estructuras no se encuentran situados en porciones de memoria secuencial, como el caso de los arreglos.

Los elementos se relacionan cada uno entre si, debido a campos enlace o punteros.

Los elementos cada uno considerado como nodo de la estructura.

Las estructuras dinámicas pueden ser:

- Unidimensionales.
- Bidimensionales.
- N-Dimensionales.
- Matrices Esparcidas.
- Tablas de Hash.

7.1.3.1. Estructuras Unidimensionales

Las estructuras unidimensionales poseen una dimensión. Los nodos apuntan cada uno solamente a una posición de memoria. Las estructuras de este tipo son:

- Pilas.
- Colas.
- Listas simples.

7.1.3.2. Estructuras Bidimensionales

Las estructuras bidimensionales poseen una representación de dos dimensiones. Los nodos cada uno apuntan hacia dos posiciones de memoria como máximo. Las estructuras de este tipo son:

- Arboles binarios no equilibrados.
- Arboles binarios equilibrados conocidos como AVL.

Los árboles binarios son estructuras bidimensionales que poseen una raíz, y que cada nodo puede contener máximo dos hijos. La raíz posee dos nodos y a partir de estos surgen las ramas del árbol. La rama izquierda puede ser del mismo tamaño, menor tamaño ó mayor tamaño que la rama derecha. La rama izquierda y la rama derecha si poseen el mismo tamaño se dice que es un árbol AVL.

7.1.3.3. Estructuras N-Dimensionales

Las estructuras de n-dimensionales hacen referencia a una cantidad N de dimensiones. Esto quiere decir que un nodo contiene más de dos direcciones de memoria. Los árboles B son una representación de este tipo de estructuras.

Los arboles B se conocen como árboles balanceados B. Los árboles B mantienen que los nodos internos deben poseer un número variable de nodos hijos dentro de un rango predefinido. Un dato al ser insertado o eliminado la cantidad de nodos hijos variará dentro de un nodo.

Un árbol b es mantenido balanceado debido a que todos los nodos hoja se encuentran a la misma altura

7.1.3.4. Matrices Esparcidas

Las matrices esparcidas se conocen como listas ortogonales. Estas listas son utilizadas para representar matrices.

Los nodos de las listas ortogonales son cuatro. Los nodos poseen cuatro punteros. Estos punteros son:

- Uno para el puntero del nodo izquierdo
- Otro para el puntero del nodo derecho
- Otro para el puntero del nodo inferior
- Otro para el puntero del nodo superior

Las listas ortogonales permiten la creación de matrices NxM.

7.1.3.5. Tablas de Hash

Las tablas de Hash realizan la asociación entre llaves (claves) con valores. La implementación de estas tablas suele ser sobre arreglos de una dimensión. Estas tablas son útiles cuando se almacenan grandes cantidades de información.

7.2. Pilas

Las pilas se conocen como *stack*. Una pila es un conjunto ordenado de elementos. Estos elementos solamente se pueden acceder por un lugar único o extremo de la pila.

Las pilas utilizan la política de acceso a datos del tipo *LIFO*. *LIFO* son las siglas para *last in first out*. Esta política indica que se posee una estructura donde los datos se pueden sacar solamente de un lado, entonces cada dato que llega se apila en la estructura y el último dato en llegar será el primero en salir. *LIFO* es comparado comúnmente con una pila de platos donde el que se

Figura 46. Páginas 145 – 148 Libro “Programación principiantes”

utiliza es el que se encuentra arriba de la pila. La figura 7.1. muestra la estructura de una pila.

Figura 7.1. Estructura de una pila

Una pila se compone de elementos que se almacenan uno sobre otro. Esto corresponde a la forma que entran en la pila. La parte superior de la pila es denominada cima. Este es el único lugar donde se pueden extraer elementos. Esto indica que cada vez solamente se puede acceder al elemento *TOS* (*top of stack*).

La clase pila debe contener los siguientes métodos:

- CrearPila.
- Poner.
- Sacar.
- Reestablecer.
- Cima.
- EsPilaLlena.
- EsPilaVacia.
- TamañoPila.

Las pilas suelen obtener importancia debido a que permiten la evaluación de expresiones en notación posfija.

Las pilas se utilizan para reconocedores sintácticos independientes del contexto y para la implementación de la recursividad.

7.2.1. Especificación de Operaciones

Las pilas manejan 4 operaciones básicas:

- Poner: esta operación permite añadir un elemento a la pila. (A la cima).
- Sacar: esta operación permite sacar un elemento de la pila.
- Cima: esta operación permite obtener el elemento en la cima de la pila.
- Restablecer: esta operación permite remover todos los elementos y dejar la pila vacia.

Las operaciones sobre las pilas deben ser capaces de manejar dos problemas que ocurren en ellas.

- Pila vacia: esta también se conoce como desbordamiento negativo (underflow). Este problema ocurre cuando se intenta sacar un elemento de la pila y ésta se encuentra vacia.
- Pila llena: esta también se conoce como desbordamiento positivo (overflow). Este problema ocurre cuando se intenta meter un elemento en una pila que se encuentra llena. Este problema generalmente ocurre cuando se manejan pilas de tamaño definido.

La operación que permite establecer el tamaño de una pila dependerá del tipo de pila. Los dos tipos de tamaño de una pila son:

- De tamaño definido, es decir que al momento de definirse la pila se establece un tamaño. La representación se da por medio de arreglos.
- De tamaño variable, es decir que no se define un tamaño para la pila. La representación se puede utilizar mediante listas enlazadas o por medio de la clase pila (clase definida en lenguajes orientados a objetos).

7.2.2. Implementación Estática

La implementación estática corresponderá a una clase que utilizará un arreglo que integrará a la pila. Esta clase debe poseer tres atributos:

- Datos: el cual será del tipo arreglo que almacenará los elementos de la pila y será del tipo de datos que almacenará el arreglo.
- Tamaño: este guardará el tamaño máximo del arreglo Datos.
- Indice: permite indicar la posición en Data del siguiente elemento a guardar.

La clase debe poseer los siguientes métodos:

- CrearPila: este método será el constructor de la clase pila y definirá el tamaño del arreglo.
- Poner: este método permitirá meter un elemento a la pila si esta no se encuentra llena.
- Sacar: este método permitirá sacar un elemento de la pila si ésta no se encuentra vacia.
- Restablecer: Este método permite vaciar la pila.

- Cima: Este método permite ver el elemento en la cima sin sacarlo.
- EsPilaLlena: permite verificar si la pila se encuentra llena, es de tipo boolean.
- EsPilaVacia: permite verificar si la pila se encuentra vacia, es de tipo boolean.
- TamañoPila: permite devolver el tamaño actual de la pila.

Los métodos tienen que ajustarse a vigilar el arreglo Datos. Estos realizan operaciones sobre este arreglo.

7.2.3. Implementación Dinámica

La implementación dinámica corresponde a realizar una clase pila. Esta clase pila corresponderá a una lista simplemente enlazada. Esto quiere decir que debe contener un nodo que realizará la función del tipo de datos. La clase pila tendrá que llevar el control del primer nodo de la lista. Esto permitirá que no se pierda la lista cuando ya posea más de un elemento.

Los métodos concernientes a esta clase tendrán que operar sobre la lista simplemente enlazada. Esto quiere decir que posee los mismos métodos que la clase pila estática solamente que en lugar de controlar un arreglo controlan la lista enlazada. El nuevo elemento insertado empezará desde la cima. Esto quiere decir que en el apuntador del nuevo elemento (que ahora es la cima) se referenciará al nodo que era considerado como cima de la pila. El sacar un elemento de la pila implica que la cima y la cabeza de la lista sean redirigidas al apuntador de la cima actual.

La otra forma de realizar esta implementación dinámica es utilizando la clase que brinda el lenguaje orientado a objetos en el que se está programando. La clase pila permite el manejo de pilas de una forma fácil. Esta clase permite crear pilas del tipo Objeto. Esto quiere decir que son pilas genéricas cuyos elementos pueden ser cualquier tipo de datos.

El objeto Pila se crea con la siguiente sintaxis:

- Pila MiPila = new Pila();

Esto permite crear una pila vacia.

Los métodos que utiliza este objeto son:

- vacia(): devuelve tipo *boolean* y devuelve si la pila se encuentra vacia.
- ver(): devuelve tipo Objecto . Mira el objeto en la cima y lo devuelve sin sacarlo de la pila.
- sacar(): devuelve tipo Objecto, remueve un objeto en la cima de la pila y regresa el objeto como valor de la función.
- meter(Objecto elemento): permite poner un elemento en la cima de la pila.
- buscar(Objecto o): este es de tipo int. Devuelve la posición donde se encuentra un elemento en la pila y -1 en caso de que no se encuentre.

7.3. Colas

7.3.1. Introducción a la Teoría de Colas

La teoria de colas es el estudio (matemático) de cómo se comportan las lineas de espera. Una linea de espera (cola) ocurre cuando un cliente solicita un servicio y el servidor se encuentra ocupado y este debe esperar.

La teoria de colas permite mediante modelos matemáticos modelar una linea de espera.

La importancia de la teoria de colas radica en:

- Permite la identificación del nivel óptimo de capacidad del sistema. Esto permite reducir costes.
- Comprobar el impacto en el coste total del sistema al realizar modificaciones a la capacidad del sistema.
- Establecer un balance entre costes (cuantitativos) y servicio (cualitativos).
- Considerar el tiempo de permanencia en la cola. Esto debido que dependiendo del tipo de servicio así será el tiempo que un cliente podría considerar esperar antes de abandonar el sistema.

7.3.2. Conceptos

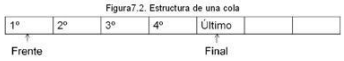
Las colas se conocen también como *queues*. Una cola es una estructura de datos que permite el almacenamiento de elementos en una lista y que permite acceder a los mismos por ambos extremos de la lista.

Figura 47. Páginas 149 – 152 Libro “Programación principiantes”

Las colas poseen una política de acceso a datos del tipo FIFO. FIFO son las siglas para first in firstout. Una cola permite que un dato sea ingresado al en el extremo final de la cola y posteriormente pueda sacarse en el extremo inicial de la cola. Esto quiere decir que el primer elemento en entrar será el primer elemento en salir.

La estructura de una cola se aprecia en la figura 7.2.

Figura7.2. Estructura de una cola



Una cola se encuentra formada por elementos que ingresan a la cola en orden y salen de ella en el mismo orden como entraron. Las colas cuentan con dos extremos:

- Frente que es por donde salen los elementos de la cola.
- Final que es por donde ingresan los elementos de la cola.

Las operaciones básicas de una cola deben ser:

- CrearCola
- Insertar
- Remover
- EsColaVacia
- EsColaLlena
- Frente
- TamañoCola

La importancia de las colas radica en su capacidad para modelar sistemas cliente/servidor, donde las personas deben esperar a ser atendidas por el servidor y conforme este atiende ellas esperan en la cola. (Teoria de Colas).

7.3.2.1. Especificación de Operaciones

Las operaciones básicas en colas son parecidas a las de pilas:

- poner: esta operación permite ingresar un elemento en el extremo final de la cola.

- sacar: esta operación permite sacar un elemento en el extremo front de la cola.
- cima: esta operación permite consultar el elemento en el frente de la cola.
- restablecer: esta operación permite vaciar la cola.

Las colas al igual que las pilas deben considerar una cola vacia y una cola llena.

Una cola vacia es considerada cuando la cola no posee elementos y la posición del frente coincide con la del final de la cola.

Una cola llena es aquella que se encuentra llena de elementos y la posición de final de la cola se encuentra en la última posición y se encuentra ocupada.

El tamaño de la cola al igual que el de las pilas puede variar. Esto es porque hay dos tipos de colas:

- Estática: esta es la cola cuyo tamaño se encuentra definido al momento de crearla (Utiliza arreglos).
- Dinámica: esta cola no posee un tamaño definido, su tamaño puede variar. (Utiliza listas enlazadas).

Las colas pueden ser manipuladas de dos formas. Esto resulta del modelo de manipulación simple y circular. Una cola puede ser creada de dos tipos:

- Simple ó lineal, es decir que es un arreglo o lista unidimensional el cual albergará los elementos y por esto se necesita 2 indices que referenciaran el frente y final de la cola.
- Circular, esta es otra implementación que permite unir el extremo final de la cola con su extremo frente.

7.3.2.2. Implementación Estática

La implementación estática de una cola se realiza mediante arreglos. La clase cola debe poseer cuatro atributos. Estos atributos son:

- Datos: el cual será del tipo arreglo que almacenará los elementos de la pila y será del tipo de datos que almacenará el arreglo.
- Tamaño: este guardará el tamaño máximo del arreglo Datos.
- Final: permite indicar la posición actual del final de la cola.

- Frente: permite indicar la posición actual del frente de la cola.

La clase cola debe poseer los siguientes métodos:

- CrearCola: este método será el constructor de la clase cola y definirá el tamaño del arreglo.
- Insertar: este método permitirá meter un elemento a la cola al final, si esta no se encuentra llena.
- Remover: este método permitirá sacar un elemento de la cola desde el frente si ésta no se encuentra vacía.
- Restablecer: Este método permite vaciar la cola.
- Frente: Este método permite ver el elemento en el frente sin sacarlo.
- EsColaLlena: permite verificar si la cola se encuentra llena, es de tipo boolean.
- EsColaVacia: permite verificar si la cola se encuentra vacía, es de tipo boolean.
- TamañoCola: permite devolver el tamaño actual de la cola.

7.3.2.3. Implementación Dinámica

La implementación dinámica de una cola puede ser realizada de dos formas.

La primera forma corresponde a una implementación de listas enlazadas. Esta implementación definirá una lista enlazada que albergará elementos del tipo que se necesite guardar. La implementación por listas permite que la cola sea especializada. La clase debe ser definida como en la clase cola estática. La consideración es que ya no se trabajará con un arreglo sino que se trabajará con listas enlazadas en las cuales no se tiene un tamaño definido desde el inicio. Los elementos en la cola que sean insertados serán colocados en el apuntador de la posición final de la cola. Esto permite que la clase sepa cuál es su final y así inserte ahí los elementos. El frente de la cola cambiará conforme se saquen los elementos de la cola. Esto hace que al sacar un elemento el frente cambie a la posición del apuntador del nodo del elemento que fue sacado de cola. Los atributos y métodos son los mismos que la de la clase estática considerando que es una lista enlazada la que se utiliza como cola y no un arreglo.

La otra forma de realizar una cola es utilizando la clase genérica cola que viene implementada generalmente en los lenguajes orientados a objetos (Java ó C#). Esta es genérica debido a que los elementos que alberga son de tipo

Objeto. La clase cola puede en algunos lenguajes (Java) ser del tipo abstracto. Esto conllevará a que se utilice como herencia de otra clase.

Los métodos que utiliza esta clase genérica son:

- *boolean* ofrecer(Objecto o): este método permite insertar un elemento en la cola, si es posible.
- Objecto sondear(): regresa y remueve el elemento en el frente de la cola ó devuelve *null* si la cola se encuentra vacía.
- Objecto ver(): retorna el elemento en el frente de la cola pero no remueve el elemento, regresa *null* si no hay elementos.
- limpiar(): elimina todos los elementos en la cola.
- *int* tamaño(): regresa el total de elementos en la cola.

7.3.3. Cola de Prioridades

Una cola de prioridades es una cola donde los elementos están asociados a una prioridad y esta determina el orden en que saldrán los elementos. El caso en que varios elementos poseen la misma prioridad permite que salgan de forma como saldrían en una cola normal.

La definición de una cola de prioridades se puede hacer de dos formas:

- Añadiendo un campo a cada elemento con su prioridad. (Debe de mantenerse la cola ordenada por orden de prioridad).
- Por cada prioridad se creará una cola. Los elementos serán almacenados debido a su prioridad en la cola correspondiente a esta prioridad.

Las operaciones serán las mismas que la de las colas normales. Los únicos métodos modificados serán el de poner y sacar, los cuales ahora deberán considerar la prioridad, dependiendo del tipo de definición de la cola de prioridad.

7.4. Listas Enlazadas

Una lista enlazada es un conjunto de elementos en el que cada elemento se encuentra uno detrás de otro. Los elementos cada uno consta de dos partes la primera sería su valor y la segunda un puntero al siguiente elemento de la lista.

Figura 48. Páginas 153 – 156 Libro “Programación principiantes”

7.4.1. Nodo

Un nodo es un elemento de la lista que posee dos campos:

- El primero su valor que puede ser cualquier tipo.
- El segundo es un puntero que apunta al siguiente elemento de la lista.

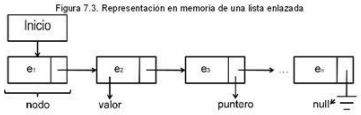
El campo de valor es del tipo de dato que se desee guardar. Esto permite que si la lista necesita guardar tipo entero el campo de valor será entero. Esto también significa que si se desea guardar estructuras compuestas por varios campos sea posible definir este valor del tipo objeto.

El campo del puntero es del tipo de la clase en que se ha definido el nodo.

7.4.2. Definición y Representación en Memoria

Una lista enlazada en memoria se verá como una secuencia de elementos uno detrás de otro, que existen en diferente parte de la memoria. La figura 7.3. muestra la disposición en memoria de una lista enlazada.

Figura 7.3. Representación en memoria de una lista enlazada



7.4.3. Clasificación

Las listas enlazadas pueden ser de cuatro tipos. Estos tipos son:

- Simplemente enlazadas.
- Doblemente enlazadas.

- Circular simplemente enlazadas.
- Circular doblemente enlazadas.

Las listas simplemente enlazadas se identifican debido a que cada nodo contiene un único enlace que lo conecta con el siguiente nodo. Estas listas se caracterizan por ser eficientes en recorridos directos (adelante).

Las listas doblemente enlazadas se encuentran compuestas por nodos que poseen dos enlaces. Un enlace es hacia su nodo sucesor y el otro enlace es hacia su nodo predecesor. Estas listas se caracterizan por ser eficientes en recorridos directos (adelante) y en recorridos inversos (atrás).

La lista circular simplemente enlazada es una lista simplemente enlazada con la única diferencia que el último nodo hace referencia al primer nodo de la lista.

La lista circular doblemente enlazada es una lista doblemente enlazada con la diferencia que el primer nodo de la lista hace referencia en su parte predecesor al último nodo de la lista. El último nodo de la lista en su parte sucesor hace referencia el primer nodo de la lista.

7.4.4. Listas Enlazadas

Las listas enlazadas se basan en dos aspectos. El primer aspecto es definir un nodo. El otro aspecto definir una clase que controlará los nodos.

El nodo será definido por una clase. Esta clase posee dos atributos. El atributo valor que será del tipo del valor que se desee guardar elementos. El atributo siguiente que será un apuntador al siguiente elemento de la lista. Este atributo es del tipo de la misma clase nodo. El constructor de la clase permite asignar valor por parámetro al atributo valor. Este también La sintaxis es:

```
public class Nodo
{
    public tipo valor;
    public Nodo siguiente;
    public Nodo (Object n)
    {
        valor = n;
        siguiente = null;
    }
```



```
    }
```

La creación de la clase nodo permite que se enfoque en la creación de la clase lista. La lista se debe definir en una clase que contendrá como atributos el primer nodo de la lista. Este nodo será la cabecera de la lista.

La clase ListaSE deberá contener los métodos para poder utilizar la lista y los atributos necesarios para poder administrar la lista.

Las operaciones básicas que debe manejar la clase ListaSE son la de creación, inserción, búsqueda y eliminación.

La creación de la lista se realiza por medio de la clase ListaSE. La sintaxis es:

```
public class ListaSE
{
    private Nodo inicio;
    public ListaSE()
    {
        inicio = null;
    }
    public void insertar(tipo E){}
    public Object buscar(tipo E){}
    public void eliminar(tipo E){}
}
```

La inserción de un elemento se contempla al inicio de la lista, al final, antes de y después de.

La sintaxis para el método de insertar al inicio de la lista es:

```
public void insertar_cabeza(tipo O)
{
    Nodo aux = inicio;
    inicio = new Nodo(O);
    inicio.siguiente = aux;
}
```

La sintaxis para el método insertar al final de la lista es:

```
public void insertar_final(tipo O)
{
```



```
    Nodo aux = inicio;
    if (aux != null)
    {
        while(aux.siguiente!=null)
        {
            aux = aux.siguiente;
        }
        aux.siguiente = new Nodo(O);
    }
    else
    {
        insertar_cabeza(O);
    }
}
```

La sintaxis para insertar antes de un elemento es:

```
public void insertar_antes (tipo O, tipo OAnterior)
{
    Nodo auxant = buscar(OAnterior);
    if (auxant!=null)
    {
        Nodo aux = inicio;
        auxant = null;
        if (aux.valor.igual(O))
        {
            return aux;
        }

        else
        {
            While ((aux.siguiente!=null) &&
(!aux.valor.igual(OAnterior)))
            {
                auxant = aux;
                aux = aux.siguiente;
            }
            if (aux.valor.igual(O))
            {
                auxant.siguiente = new Nodo(O);
                auxant = auxant.siguiente;
                auxant.siguiente = aux;
            }
```

Figura 49. Páginas 157 – 160 Libro “Programación principiantes”


```
            }
        }
    }
```

La sintaxis para insertar después de un elemento es la siguiente:

```
public void insertar_despues (tipo O, tipo OPosterior)
{
    Nodo aux = buscar(OPosterior);
    if (aux!=null)
    {
            Nodo auxsig = aux.siguiente;
            aux.siguiente = new Nodo(O);
            aux = aux.siguiente;
            aux.siguiente = auxsig;
    }
}
```

La sintaxis para buscar un elemento dentro de la lista es la siguiente:

```
public Nodo buscar(tipo O)
{
    Nodo aux = inicio;
    if (aux != null)
    {
        if (aux.valor.igual(O))
        {
            return aux;
        }
        else
        {
            While ((aux.siguiente!=null) && (!aux.valor.igual(O)))
            {
                aux = aux.siguiente;
            }
            if (aux.valor.igual(O))
            {
                return aux;
            }
            else
            {
                return null;
            }
```



```
            }
        }
        else
        {
            return null;
        }
    }
```

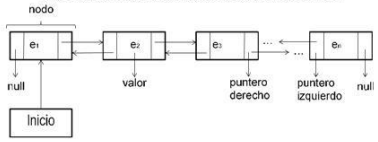
La sintaxis para eliminar un elemento en la lista es la siguiente:

```
public void eliminar(tipo O)
{
    Nodo aux = inicio;
    Nodo auxant = null;
    if (aux != null)
    {
        if (aux.valor.igual(O))
        {
            inicio = aux.siguiente;
        }
        else
        {
            While ((aux.siguiente!=null) && (!aux.valor.igual(O)))
            {
                auxant = aux;
                aux = aux.siguiente;
            }
            if (aux.valor.igual(O))
            {
                if (aux.siguiente == null)
                {
                    auxant.siguiente = null;
                }
                else
                {
                    auxant.siguiente = aux.siguiente;
                }
            }
        }
    }
```


7.4.5. Listas Doblemente Enlazadas

Las listas doblemente enlazadas se caracterizan porque poseen un nodo que apunta hacia el siguiente nodo de la lista y también un nodo que apunta al nodo anterior a la lista. La figura 7.4. muestra la representación en memoria de estas listas.

Figura 7.4. Representación de memoria de una lista doblemente enlazada



El nodo de este tipo de listas posee dos atributos tipo nodo. Estos atributos corresponden al nodo siguiente y al anterior. Esta clase nodo utiliza también un atributo valor. La sintaxis corresponde a:

```
public class NodoBinario
{
    public tipo valor;
    public Nodo siguiente;
    public Nodo anterior;
    public Nodo (tipo n){
        valor = n;
        siguiente = null;
        anterior = null;
    }
}
```

Las listas doblemente enlazadas permiten acceder a los elementos de un nodo en cualquier orden. Un puntero apunta al siguiente elemento de la lista y el otro apunta al elemento anterior.

La inserción de un elemento puede ocurrir al inicio de la lista, al fina, antes de un elemento o después de un elemento.

La inserción generalmente será igual que como se realiza con una lista enlazada simple, con la única diferencia que se debe recordar referenciar las partes anterior de los nodos implicados.

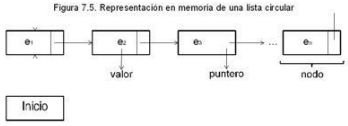
La eliminación debe de poder eliminar el nodo y a los nodos anterior y posterior debe poder enlazarlos nuevamente en su parte siguiente y anterior.

La búsqueda se puede realizar como una búsqueda de lista enlazada simple, con la diferencia que puede ser realizada tomando el final de la lista como inicio.

7.4.6. Listas Circulares

Las listas circulares se caracterizan por ser listas simplemente enlazadas en la cual el último nodo en su parte siguiente referencia a la cabeza de la lista. La figura 7.5. muestra la representación en memoria de estas listas.

Figura 7.5. Representación en memoria de una lista circular

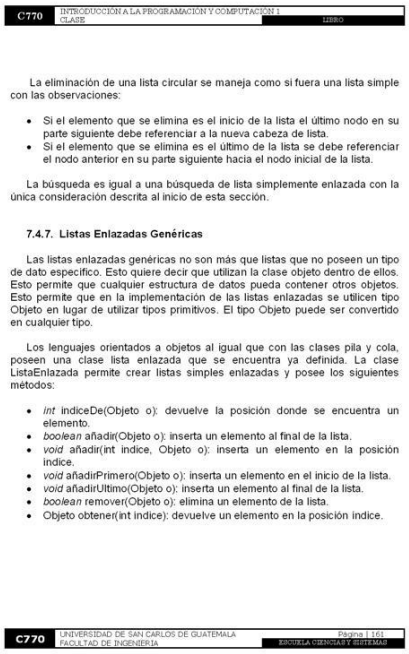


El nodo de la lista circular es la misma definición del nodo de la lista simplemente enlazada.

Las operaciones que necesiten recorrer la lista circular deben considerar que la lista se ha recorrido una vez si empezando desde el inicio de la lista se ha encontrado que un nodo en su parte siguiente es igual al nodo inicial de la lista.

La inserción de un elemento en una lista circular debe considerar si se está insertando antes del inicio de la lista, para lo cual deberá cambiar el apuntador al inicio de la lista y ser sustituido por el nuevo nodo.

Figura 50. Página 161 Libro “Programación principiantes”



La eliminación de una lista circular se maneja como si fuera una lista simple con las observaciones:

- Si el elemento que se elimina es el inicio de la lista el último nodo en su parte siguiente debe referenciar a la nueva cabeza de lista.
- Si el elemento que se elimina es el último de la lista se debe referenciar el nodo anterior en su parte siguiente hacia el nodo inicial de la lista.

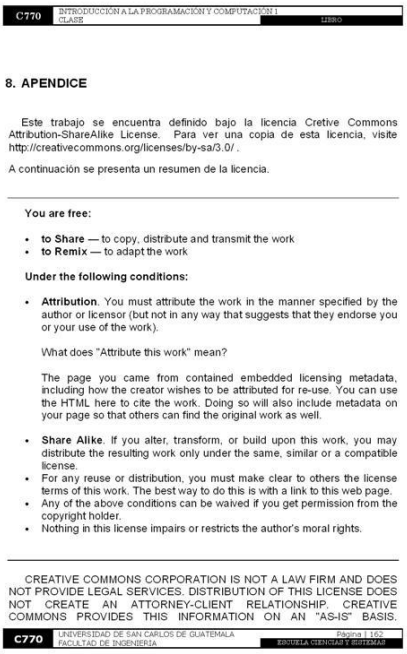
La búsqueda es igual a una búsqueda de lista simplemente enlazada con la única consideración descrita al inicio de esta sección.

7.4.7. Listas Enlazadas Genéricas

Las listas enlazadas genéricas no son más que listas que no poseen un tipo de dato específico. Esto quiere decir que utilizan la clase objeto dentro de ellos. Esto permite que cualquier estructura de datos pueda contener otros objetos. Esto permite que en la implementación de las listas enlazadas se utilicen tipo Objeto en lugar de utilizar tipos primitivos. El tipo Objeto puede ser convertido en cualquier tipo.

Los lenguajes orientados a objetos al igual que con las clases pila y cola, poseen una clase lista enlazada que se encuentra ya definida. La clase ListaEnlazada permite crear listas simples enlazadas y posee los siguientes métodos:

- *int* indiceDe(Objeto o): devuelve la posición donde se encuentra un elemento.
- *boolean* añadir(Objeto o): inserta un elemento al final de la lista.
- *void* añadir(int indice, Objeto o): inserta un elemento en la posición indice.
- *void* añadirPrimero(Objeto o): inserta un elemento en el inicio de la lista.
- *void* añadirUltimo(Objeto o): inserta un elemento al final de la lista.
- *boolean* remover(Objeto o): elimina un elemento de la lista.
- Objeto obtener(int indice): devuelve un elemento en la posición indice.



8. APENDICE

7. DOCUMENTACIÓN DE APOYO DEL CURSO DE INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2

7.1. Descripción

Documentación de apoyo del curso de IPC2, el cual fue realizado en formato de un libro que contiene las unidades y temas más importantes impartidos en el curso.

El libro consta de cinco capítulos. Estos capítulos son:

a. Introducción a las bases de datos relacionales.
b. Metodología para desarrollo de *software*.
c. Etapa de análisis del ciclo de construcción.
d. Etapa de diseño del ciclo de construcción.
e. Integración de artefactos de *software*.

Figura 51. Páginas 1 – 4 Libro “Programación intermedios”

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

ESCUELA DE CIENCIAS Y SISTEMAS

LIBRO

NOMBRE DE CURSO:	Introducción a la Programación y Computación 2 CLASE
CODIGO DEL CURSO:	0771
NOMBRE DEL MODULO:	Programación Intermedios
AUTOR:	Mario José Bautista Fuentes

CONTENIDO

1. INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES

1.1. Conceptos Básicos de Bases de Datos

1.1.1. Modelo

La representación de algo que existe en la vida real es básicamente un modelo. Un modelo es una abstracción simplificada de la realidad.

1.1.2. Base de Datos

Una base de datos es un conjunto de datos relacionados entre sí y que reúne dos condiciones:

- Ser persistentes, es decir que deben ser almacenados para poder ser utilizados posteriormente.
- Ser utilizados por una aplicación de software. Esto indica que los datos deben estar de tal forma que una aplicación puede interpretarlos y realizar operaciones con los datos.

1.1.3. Sistema Administrador de Base de Datos

El sistema administrador de base de datos es una herramienta de tecnología que soporta a la base de datos. Las siglas en ingles son DBMS (*data base management system*). Actualmente los DBMS más importantes son:

- Oracle
- SQL Server
- My SQL
- PostgreSQL

1.1.3.1. Usuario

Un usuario es la persona que utiliza las aplicaciones y la base de datos. El usuario puede ser:

- Programador de la aplicación: este usuario crea la aplicación que permite hacer uso de la base de datos. Este usuario no es el encargado de la base de datos.
- Usuario final: este usuario hace uso de la aplicación. El usuario final no comprende cómo la aplicación interactúa con la base de datos. Este usuario solamente utiliza la aplicación.
- DBA (administrador de base de datos): este usuario administra la plataforma electrónica de la base de datos. El DBA es el responsable de los datos que se almacenan en la base de datos y de la estructura que estos poseen.

1.1.3.2. Arquitectura de 3 Capas

La arquitectura de capas es un estilo de programación, cuyo fin es separar la lógica del negocio de la lógica de diseño.

Una capa debe entenderse como la manera en que una solución es dividida (segmentada) desde un punto de vista lógico. Esto permite diferenciar a una capa de un nivel. Este último es como se encuentran divididas las capas de una manera física.

La arquitectura de 3 capas realiza la separación identificando 3 capas. Las capas se pueden encontrar en una misma computadora o pueden ser divididas en varias. Las capas de esta arquitectura son:

- Capa de Presentación: esta capa algunas veces se le puede denominar como capa de usuario. La capa de presentación permite presentar el sistema al usuario, le brinda información y obtiene la información del usuario. Esta capa se comunica solamente con la capa del negocio. Esta capa es visible por el usuario final y por el programador.
- Capa del Negocio: esta capa contiene los programas que se deben ejecutar. La capa del negocio obtiene las peticiones de usuario y envía las respuestas de estas peticiones. Esta capa contiene la lógica del negocio y por tal contiene todas las reglas que deben ser cumplidas.

Figura 52. Páginas 5 – 8 Libro “Programación intermedios”

Esta capa se comunica con la capa de presentación y con la capa de datos. Esta capa es visible por el programador y por el DBMS.

- Capa de Datos: en esta capa se encuentran almacenados los datos. La capa de datos se encuentra integrada por uno o más DBMS. Estos se encargan de almacenar los datos, recibir las solicitudes de almacenamiento ó recuperar la información desde la capa del negocio. Esta capa es visible por el DBMS y por el DBA.

1.1.4. Base de Datos Relacional

Una base de datos relacional es una base de datos (DB), la cual se fundamenta en el modelo relacional. Estas bases de datos son percibidas como si fueran un conjunto de tablas.

1.1.4.1. Modelo Relacional

El modelo relacional fue creado por Edgar F. Codd. Este modelo permite que los datos se estructuren a nivel lógico como tablas formadas por filas y columnas. Esta estructura a nivel fisico es completamente diferente.

El modelo relacional maneja 3 aspectos:

- La estructura de datos, esto significa que los datos en el modelo relacional se encuentran estructurados y esto garantiza su acceso.
- La integridad de datos signifca que los datos en la base de datos deben ser correctos. Los datos dentro de la base de datos harán referencias a datos existentes y en ningún momento referenciarán datos falsos o inexistentes. Asimismo ninguna tupla en la base de datos serán repetidos.
- El manejo de datos, significa el conjunto de operaciones que permiten manipular los datos.

1.1.4.2. Entidad

Una entidad representa cualquier objeto distinguible en un modelo de negocios que se debe representar en la base de datos.

Una entidad se encuentra representada en la Figura 1.1. En esta figura se presenta la notación.

Figura 1.1. Entidad

1.1.4.3. Atributos

Los atributos son la unidad fundamental que describe un dato. Una entidad se encuentra integrada por varios atributos. Existen varios tipos de datos para los atributos, entre estos:

- Numérico.
- Hilera (*String*).
- Fecha.

Las clases de atributos pueden ser:

- Obligatorio (*NOT NULL*, *).
- Opcional (*NULL*, °).

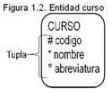
1.1.4.4. Tupla

Una tupla es considerada como el conjunto de atributos que describen una entidad.

Ejemplo 1.1.

Se desea crear una entidad que corresponda a un curso y establecer sus atributos. Para realizar esta entidad, se identifica que la tupla de la entidad se encuentra compuesta por tres atributos: código, nombre y abreviatura. La Figura 1.2. indica la notación de la entidad curso y de la tupla.

Figura 1.2. Entidad curso



La Figura 1.3. representa como se veria la entidad curso desde el punto de vista fisico y no lógico. En la representación fisica cada fila de la tabla representa una tupla y cada tupla corresponde a una serie de datos especificos y únicos. Estos datos como tupla no pueden ser iguales debido a que se estaria violando la integridad de los datos. Esto debido a que habria datos repetidos. El modelo relacional no concibe que en ningún momento en las entidades se puedan albergar tuplas repetidas.

Figura 1.3. Representación fisica de la entidad curso

CURSO

codigo	nombre	abreviatura
0771	Progra 2	IPC2
0772	Estructuras	ED

Tupla

Ejemplo 1.2.

El dueño de un hotel necesita guardar información básica relacionada con sus clientes. Él desea crear una entidad que satisfaga esta necesidad y establezca sus atributos.

La entidad necesita un nombre. Este nombre debe identificar adecuadamente a la entidad. El nombre será huésped. Los atributos para esta entidad serán: código del huésped, nombre, cedula, dirección, teléfono. La Figura 1.4. indica la notación de la entidad huésped.

Figura 1.4. Entidad huésped

La Figura 1.5. representa la entidad huésped desde el punto de vista fisico.

Figura 1.5. Representación fisica de la entidad huesped

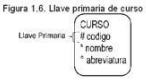
HUESPED

codigo	cedula	nombre	direccion	telefono
1231	A1-27454	Juan Pérez	Zona 1	23454433
5345	A1-44344	José Soza	Zona 2	24453444

1.1.4.5. Llaves Primarias

Una llave primaria es el identificador único para cada tupla de una entidad. La llave primaria puede ser uno o más atributos que hacen única la tupla. Estos atributos son obligatorios. Esta llave identifica a una tupla y la hace única de las demás tuplas dentro de una entidad. La Figura 1.6. presenta la notación y especifica la llave primaria dentro de una entidad.

Figura 1.6. Llave primaria de curso



1.1.4.6. Llaves Foráneas

Una llave foránea es el conjunto de uno o más atributos que permiten relacionar a la entidad a la que pertenece con la entidad que se relaciona la misma. Las llaves foráneas son referencias de una entidad hacia otras.

Ejemplo 1.3.

Dos entidades, entre ellas estudiante y curso se relacionan de tal forma que un estudiante pude asignarse varios cursos. Asimismo un curso puede ser asignado por un estudiante varias veces con el transcurso de los semestres. La Figura 1.7. representa la relación asignación entre las entidades estudiante y curso. La identificación de las tuplas para cada entidad es lo más importante. Esto permite que se identifique para la entidad asignación dos atributos. Estos atributos forman parte de la tupla de la entidad asignación. Los atributos corresponden a la llave foránea de la entidad estudiante y la llave foránea de la

Figura 53. Páginas 9 – 12 Libro “Programación intermedios”

entidad curso. Estas llaves foráneas hacen referencia a las llaves primarias de las entidades citadas. La tupla de la entidad asignación se encuentra compuesta por:

- no asignacion
- estado
- semestre
- carnet (llave foránea de entidad estudiante)
- codigo (llave foránea de entidad curso)

Figura 1.7. Relación asignación

1.1.5. Diagrama Entidad Relación

El diagrama entidad relación es la manera de expresar gráficamente un modelo de datos. Este diagrama se encuentra compuesto de entidades y las relaciones entre las mismas.

1.1.5.1. Reglas de Entidades

El diseño de las entidades implica una serie de reglas. Las reglas para el diseño de entidades son:

- Utilizar una caja con los bordes redondeados (*softbox*). Esta representará a la entidad.
- El nombre distintivo de la entidad debe estar escrito en singular y en mayúsculas. Esto es debido a que lógicamente una entidad es un objeto, aunque físicamente pude contener una serie de datos dados en tuplas.

1.1.5.2. Reglas de Atributos

El diseño de los atributos de las entidades implica una serie de reglas. Las reglas para el diseño de atributos son:

- El nombre debe encontrarse en singular y en minúsculas.
- El nombre de la entidad no puede ser el nombre del atributo.
- El atributo cuando es obligatorio se marca con el simbolo *.
- El atributo cuando es opcional se marca con el simbolo °.
- La llave primaria se marca con el simbolo #.
- Las llaves alternas se marcan con el simbolo (#).

1.1.6. Relaciones Básicas entre Entidades

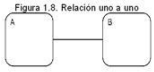
Una relación es aquella que permite describir una correspondencia entre los datos. Estas surgen debido a que una entidad puede relacionarse con una o varias entidades. Las relaciones pueden ser:

- De uno a uno.
- De uno a muchos.
- De muchos a muchos.

1.1.6.1. De Uno a Uno

La relación de uno a uno ocurre cuando una instancia en la entidad A se encuentra relacionada únicamente a una instancia de la entidad B. La Figura 1.8. presenta la notación de una relación de este tipo.

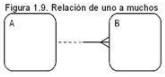
Figura 1.8. Relación uno a uno



1.1.6.2. De Uno a Muchos

Una relación de uno a muchos ocurre cuando una instancia en la entidad A se encuentra relacionada con varias instancias en la entidad B. La Figura 1.9. representa la notación de una entidad de este tipo.

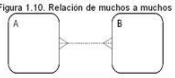
Figura 1.9. Relación de uno a muchos



1.1.6.3. De Muchos a Muchos

Una relación de muchos a muchos ocurre cuando una instancia en la entidad A se encuentra relacionada con varias instancias en la entidad B. La Figura 1.10. representa la notación de una entidad de este tipo.

Figura 1.10. Relación de muchos a muchos



1.1.6.4. Opcionalidad en las Relaciones

Las relaciones pueden ser opcionales u obligatorias. Esto es debido a que en la tupla algunos datos siempre deben aparecer como es el caso de la llave primaria.

Las relaciones opcionales se representan por medio de líneas punteadas. La figura 1.11. representa la notación de una relación opcional.

Figura 1.11. Relación opcional

Las relaciones obligatorias se representan por medio de líneas continuas. La Figura 1.12. representa la notación de una relación obligatoria.

Figura 1.12. Relación obligatoria

Las relaciones de uno a muchos (1:M) pueden ser de dos tipos:

- La Figura 1.13. presenta las relaciones más comunes de 1:M.

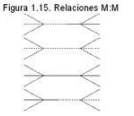
Figura 1.13. Relaciones Tipo 1, 1:M

- La Figura 1.14. presenta las relaciones menos comunes (si se tiene una de estas se recomienda revisar el modelo e intentar eliminarlas):

Figura 1.14. Relaciones tipo 2, 1:M

Generalmente las relaciones de muchos a muchos (M:M), surgen inicialmente en un modelo pero deben de ser eliminadas. Estas relaciones posteriormente dan paso a relaciones 1:M. La figura 1.15. presenta la notación de relaciones M:M.

Figura 1.15. Relaciones M:M



1.1.6.5. Relaciones como UID

UID significa *unique identificator*. Las relaciones como UID se fundamentan en que un campo de la entidad A forma parte de la llave primaria de la entidad B. La Figura 1.16. representa la notación de una relación UID.

Figura 1.16. Relación UID

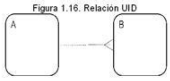


Figura 54. Páginas 13 – 16 Libro “Programación intermedios”

1.1.6.6. Eliminar Relación Muchos a Muchos

Las relaciones de M:M surgen en un principio pero estas deben ser eliminadas y sustituidas por relaciones de 1:M.

Ejemplo 1.4.

Se posee la siguiente relación entre la entidad estudiantes y la entidad curso. La figura 1.17. representa esta relación. Un estudiante se puede asignar varios cursos y un curso puede ser asignado por varios estudiantes.

Figura 1.17. Ejemplo relación M:M

Esta relación se elimina añadiendo una nueva entidad entre la entidad estudiante y la entidad curso. Estas deben ser relacionadas 1:M con la entidad nueva.

La Figura 1.7. representa la relación de asignación. Esta figura presenta como debería de quedar la relación entre la entidad estudiante y la entidad curso después de eliminar la relación M:M.

La entidad asignación se relaciona a la entidad estudiante y a la entidad curso. Esto permite indicar que un estudiante puede asignarse varias veces un curso y un curso puede ser asignado varias veces por un estudiante.

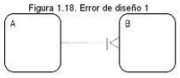
1.1.6.7. Errores de Diseño

Cuando se realiza el diseño de un diagrama entidad relación pueden ocurrir errores de diseño. A continuación se listan algunos errores muy frecuentes.

La relación no puede ser opcional del lado de muchos. Este error se encuentra representado en la Figura 1.18. Este error ocurre debido a que la relación de muchos no puede ser opcional. Esto es porque para que una tupla en la entidad A no necesariamente posee una relación con la entidad B. La

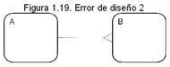
tupla en la entidad B existe debido a que se encuentra relacionada con alguna tupla de la entidad A. Esta tupla en B si no se encuentra relacionada con la tupla en A no debe de existir. Esto conlleva a indicara que del lado de muchos debe ser obligatorio.

Figura 1.18. Error de diseño 1



La barra se coloca siempre en el extremo de muchos. Este error se encuentra representado en la Figura 1.19. debido a que la barra se encuentra en el lado de uno. La barra permite indicar que la llave primaria de la entidad A formará parte de la entidad B. Esto no puede determinarse debido a que del lado de uno no existe llave foránea. Esto cambia del lado de muchos debido a que en esta entidad si existe llave foránea referente a la entidad A.

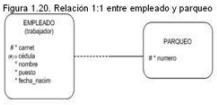
Figura 1.19. Error de diseño 2



Ejemplo 1.5.

La empresa TATA posee un sistema de parqueos para sus empleados. Este sistema permite asociar a cada uno de los empleados un parqueo siempre. El empleado siempre poseerá su parqueo y nunca lo perderá a menos que sea despedido de la empresa. El enunciado permite identificar dos tipos de entidades. La entidad empleado y la entidad parqueo. Esto permite establecer la relación de uno a uno entre estas entidades. La opcionalidad parte en que un parqueo no aparece a menos que un empleado esté asociado. Esto permite indicar que un empleado puede poseer un parqueo y que un parqueo debe pertenecer a un empleado. La entidad empleado posee los atributos: carnet, cédula, nombre, puesto, fecha de nacimiento. La entidad parquero posee únicamente el número del parqueo. La figura 1.20. representa esta relación.

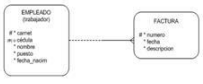
Figura 1.20. Relación 1:1 entre empleado y parqueo



Ejemplo 1.6.

Una empresa de productos posee una facturación en la cual un empleado puede realizar una o más facturas durante el día. Esto permite establecer que las dos entidades sean empleado y factura. La relación existente entre estas entidades es de uno a muchos. La opcionalidad de esta relación es opcional del lado de uno y totalmente obligatoria de lado de muchos. Esto permite indicar que un empleado puede realizar una o más facturas pero una o más facturas deben ser hechas por solamente un empleado. Los atributos de la entidad empleado son: canet, cédula, nombre, puesto, fecha de nacimiento. Los atributos de la entidad factura son: número de factura, fecha, descripción. La figura 1.21. representa la relación de 1:M.

Figura 1.21. Relación 1:M entre empleado y factura



Ejemplo 1.7.

La atención de clientes en una empresa es primordial. Los clientes llegan a la empresa y pueden ser atendidos por uno o más empleados. Los empleados también pueden atender uno o más clientes. La relación permite identificar las entidades empleado y cliente. La relación que surge entre estas entidades es de muchos a muchos. Los atributos de la entidad empleado son: carnet, cédula, nombre, puesto y fecha de nacimiento. La entidad cliente posee los atributos: número de cliente, nombre, dirección y teléfono. La figura 1.22. representa este tipo de relaciones.

Figura 1.22. Relación M:M entre empleado y entidad

La relación existente en la figura 1.22. debe de desaparecer. Esto ocasiona que se deba crear una nueva relación entre empleado y cliente. La nueva entidad únicamente será utilizada para poder llevar el control del historial de atención de un empleado con un cliente. Esta entidad se denominará atención. Los atributos de esta entidad son: fecha de atención, descripción y las llaves foráneas de las entidades empleado y cliente.

Figura 1.23. Quitando la relación M:M entre empleado y cliente

1.1.7. Diseño de Base de Datos

1.1.7.1. Mapeo Conceptual

El mapeo conceptual se realiza cuando se posee un diagrama o modelo de datos finalizado.

El mapeo conceptual se caracteriza por ser independiente del DBMS. Esta independencia lo vuelve en cierto grado genérico.

El mapeo conceptual es importante que se realice antes de que se realice el esquema conceptual. La Tabla 1.1. es la representación lógica y física de cómo se observan las entidades.

Tabla 1.1. Representación lógica y física

Lógica	Física
Entidades	Tablas
Atributos	Campos

Figura 55. Páginas 17 – 20 Libro “Programación intermedios”

Tuplas	Registros
PK	Restricciones (*Constraints*)
FK	

El mapeo conceptual se realiza por medio de definir una tabla con cuatro filas. Esta tabla se realiza una vez por cada entidad que contenga el diagrama o modelo de datos. Las filas de esta tabla representan:

- Columna: esta representa el nombre del atributo.
- Tipo de llave: representa si el atributo es llave única (UK), llave primaria (PK) ó llave foránea (FK). Esta fila pude ser vacía si el atributo no es ninguna de las llaves indicadas anteriormente.
- Opcionalidad: se coloca un *NULL* (°) o *NOT NULL* (*) dependiendo de si el atributo es opcional u obligatorio.
- Datos de Entrada: se recomienda colocar 3 ejemplos para cada atributo.

El número de columnas está dado por cada atributo que posea la entidad. En caso se poseer llaves foráneas se añade una columna por cada llave foránea que posea la entidad.

Ejemplo 1.8.

A partir de la figura 1.7. se desea mapear la entidad asignación. Esta entidad posee dos llaves foráneas.

La entidad asignación al ser mapeadas quedaría conforme a la Tabla 1.2. Esta tabla presenta el mapeo conceptual de la entidad asignación.

Tabla 1.2. Tabla de mapeo de la entidad asignación

Columna	no asignación	estado	semestre	carnet estudiante	codigo curso
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL
Columna	PK			FK2	FK1
Dato de Muestra	1	A	200801	200312942	0771
	2	A	200802	200312942	0772
	3	A	200801	200312954	0771

1.1.7.2. Esquema Conceptual

El esquema conceptual es el paso de transformar las tablas obtenidas en el mapeo conceptual a instrucciones SQL. Este paso permite obtener el script de creación de las tablas.

El nombre de las tablas debe ser escrito en plural, debido a que guardan instancias de la entidad. Las llaves primarias se recomienda que sean de tipo numérico.

La instrucción para crear tablas en SQL es “*CREATE TABLE*”. Todas las instrucciones son guardadas en un archivo de texto, que puede ser nombrado con extensión .txt ó .sql.

El tipo de los atributos queda sujeto a los soportados por el DBMS que se esté utilizando.

Ejemplo 1.9.

Se desea crear el esquema conceptual de la entidad asignación de la Figura 1.7.

A partir del mapeo conceptual de la Tabla 1.2., se debe realizar la construcción del script (esquema conceptual). El script contiene todas las instrucciones correspondientes a SQL. El tipo de los campos dependerá del DBMS utilizado. A continuación se presenta el script tomando en cuenta como DBMS a Oracle.

La entidad se crea como tabla con las palabras reservadas *CREATE TABLE*. El nombre de la tabla corresponde al nombre de la entidad. Los paréntesis indican que dentro de ellos se deben colocar los campos correspondientes a la tupla de la entidad. El script de la tabla asignación correspondería a la Figura 1.24.

Figura 1.24. Script de creación de la tabla asignación

```
CREATE TABLE asignacion (
        no_asignacion NUMERIC(7,0) NOT NULL,
        estado VARCHAR2(1) NOT NULL,
        semestre VARCHAR2(6) NOT NULL,
        carnet_estudiante NUMERIC(9,0) NOT NULL,
        codigo_curso NUMERIC(4,0) NOT NULL
    );
```

El script anterior contiene solamente la creación de la tabla asignación y de sus campos. Hasta este momento no se han establecido las *constraints*. Ahora se procede a la creación de las restricciones de la entidad asignación. La creación de estos *constraints* se realizan con las palabras reservadas *ALTER TABLE, ADD CONSTRAINT, PRIMARY KEY, FOREIGN KEY* y *REFERENCES*. La Figura 1.21. presenta el script para añadir los *constraints* a la tabla asignación.

Figura 1.25. Script de *constraints* de la tabla asignación

```
ALTER TABLE asignacion ADD CONSTRAINT
        pk_asignacion PRIMARY KEY(no_asignacion);
ALTER TABLE asignacion ADD CONSTRAINT
        fk1_asignacion  FOREIGN KEY (carnet_estudiante)
                REFERENCES estudiante (carnet);
ALTER TABLE asignacion ADD CONSTRAINT
        fk2_asignacion  FOREIGN KEY (codigo_curso)
                REFERENCES curso (codigo);
```


2. METODOLOGÍA PARA DESARROLLO DE SOFTWARE

2.1. Métodos de Programación

2.1.1. Programación Procedural

La programación procedural permite la resolución de un problema mediante la relación entre funciones.

La programación procedural posee las siguientes características:

- Se basa en la estrategia divide y vencerás (*divide and conquer*).
- Se fundamenta en condicionales, ciclos y procedimientos o funciones.
- Por su forma como se estructura, se conoce también como programación por bloques.
- Las entidades de mayor nivel de abstracción son estructuras o uniones.

2.1.2. Programación Orientada a Objetos

La programación orientada a objetos permite la resolución de un problema mediante la especificación de un objeto.

La programación orientada a objetos posee las siguientes características:

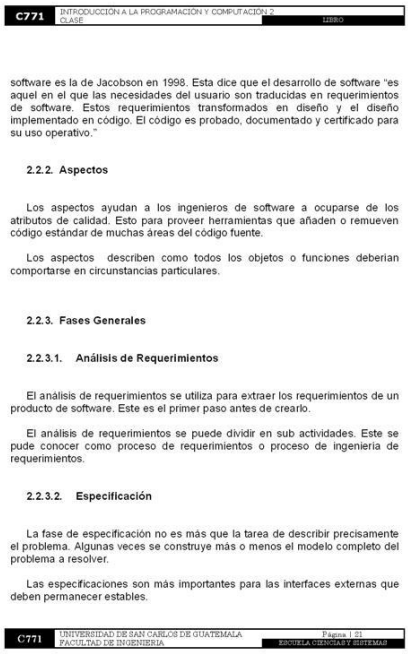
- Permite mayor encapsulamiento de los datos y operaciones.
- Permite la herencia.
- Permite el uso de sobrecarga.
- Permite polimorfismo.
- Posee mejor mantenimiento.
- Promueve componentes reutilizables.

2.2. Introducción al Proceso de Desarrollo de Software

2.2.1. Desarrollo de Software

El desarrollo de software se ha definido de diversas formas en el transcurso del tiempo. Una de las definiciones que engloba la realidad del desarrollo de

Figura 56. Páginas 21 – 24 Libro “Programación intermedios”



software es la de Jacobson en 1998. Esta dice que el desarrollo de software “es aquel en el que las necesidades del usuario son traducidas en requerimientos de software. Estos requerimientos transformados en diseño y el diseño implementado en código. El código es probado, documentado y certificado para su uso operativo.”

2.2.2. Aspectos

Los aspectos ayudan a los ingenieros de software a ocuparse de los atributos de calidad. Esto para proveer herramientas que añaden o remueven código estándar de muchas áreas del código fuente.

Los aspectos describen como todos los objetos o funciones deberían comportarse en circunstancias particulares.

2.2.3. Fases Generales

2.2.3.1. Análisis de Requerimientos

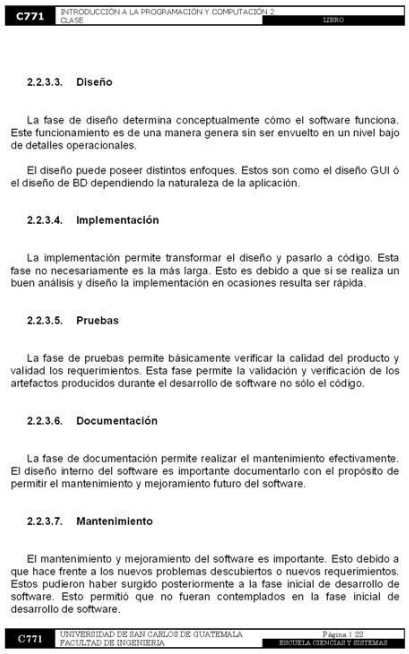
El análisis de requerimientos se utiliza para extraer los requerimientos de un producto de software. Este es el primer paso antes de crearlo.

El análisis de requerimientos se puede dividir en sub actividades. Este se pude conocer como proceso de requerimientos o proceso de ingeniería de requerimientos.

2.2.3.2. Especificación

La fase de especificación no es más que la tarea de describir precisamente el problema. Algunas veces se construye más o menos el modelo completo del problema a resolver.

Las especificaciones son más importantes para las interfaces externas que deben permanecer estables.



2.2.3.3. Diseño

La fase de diseño determina conceptualmente cómo el software funciona. Este funcionamiento es de una manera genera sin ser envuelto en un nivel bajo de detalles operacionales.

El diseño puede poseer distintos enfoques. Estos son como el diseño GUI ó el diseño de BD dependiendo la naturaleza de la aplicación.

2.2.3.4. Implementación

La implementación permite transformar el diseño y pasarlo a código. Esta fase no necesariamente es la más larga. Esto es debido a que si se realiza un buen análisis y diseño la implementación en ocasiones resulta ser rápida.

2.2.3.5. Pruebas

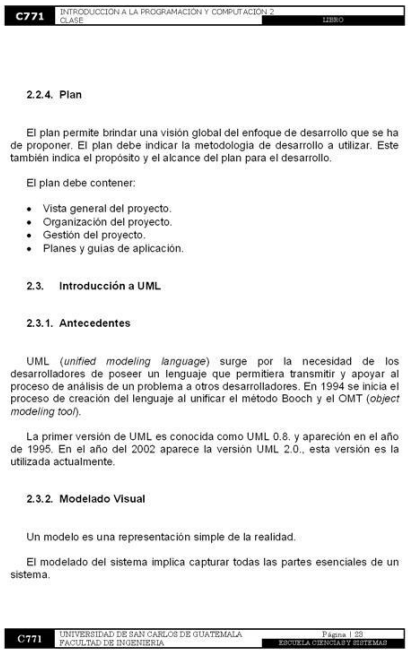
La fase de pruebas permite básicamente verificar la calidad del producto y validad los requerimientos. Esta fase permite la validación y verificación de los artefactos producidos durante el desarrollo de software no sólo el código.

2.2.3.6. Documentación

La fase de documentación permite realizar el mantenimiento efectivamente. El diseño interno del software es importante documentarlo con el propósito de permitir el mantenimiento y mejoramiento futuro del software.

2.2.3.7. Mantenimiento

El mantenimiento y mejoramiento del software es importante. Esto debido a que hace frente a los nuevos problemas descubiertos o nuevos requerimientos. Estos pudieron haber surgido posteriormente a la fase inicial de desarrollo de software. Esto permitió que no fueran contemplados en la fase inicial de desarrollo de software.



2.2.4. Plan

El plan permite brindar una visión global del enfoque de desarrollo que se ha de proponer. El plan debe indicar la metodología de desarrollo a utilizar. Este también indica el propósito y el alcance del plan para el desarrollo.

El plan debe contener:

- Vista general del proyecto.
- Organización del proyecto.
- Gestión del proyecto.
- Planes y guías de aplicación.

2.3. Introducción a UML

2.3.1. Antecedentes

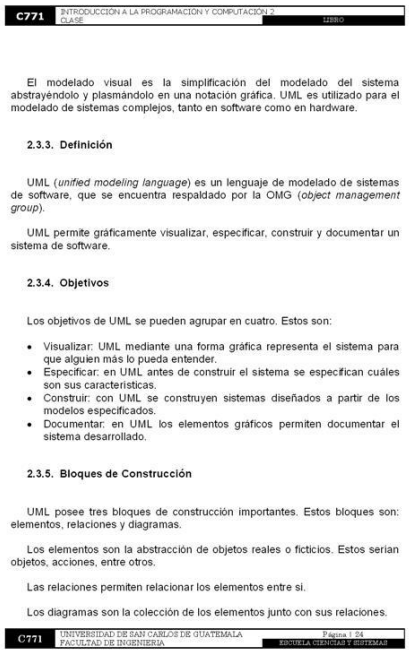
UML (*unified modeling language*) surge por la necesidad de los desarrolladores de poseer un lenguaje que permitiera transmitir y apoyar al proceso de análisis de un problema a otros desarrolladores. En 1994 se inicia el proceso de creación del lenguaje al unificar el método Booch y el OMT (*object modeling tool*).

La primer versión de UML es conocida como UML 0.8. y apareción en el año de 1995. En el año del 2002 aparece la versión UML 2.0., esta versión es la utilizada actualmente.

2.3.2. Modelado Visual

Un modelo es una representación simple de la realidad.

El modelado del sistema implica capturar todas las partes esenciales de un sistema.



El modelado visual es la simplificación del modelado del sistema abstrayéndolo y plasmándolo en una notación gráfica. UML es utilizado para el modelado de sistemas complejos, tanto en software como en hardware.

2.3.3. Definición

UML (*unified modeling language*) es un lenguaje de modelado de sistemas de software, que se encuentra respaldado por la OMG (*object management group*).

UML permite gráficamente visualizar, especificar, construir y documentar un sistema de software.

2.3.4. Objetivos

Los objetivos de UML se pueden agrupar en cuatro. Estos son:

- Visualizar: UML mediante una forma gráfica representa el sistema para que alguien más lo pueda entender.
- Especificar: en UML antes de construir el sistema se especifican cuáles son sus características.
- Construir: con UML se construyen sistemas diseñados a partir de los modelos especificados.
- Documentar: en UML los elementos gráficos permiten documentar el sistema desarrollado.

2.3.5. Bloques de Construcción

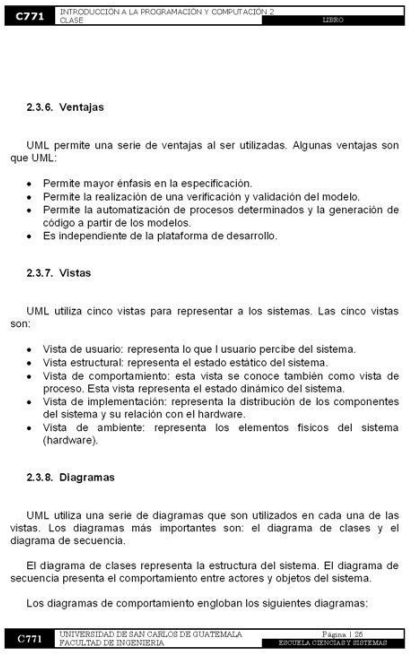
UML posee tres bloques de construcción importantes. Estos bloques son: elementos, relaciones y diagramas.

Los elementos son la abstracción de objetos reales o ficticios. Estos serían objetos, acciones, entre otros.

Las relaciones permiten relacionar los elementos entre si.

Los diagramas son la colección de los elementos junto con sus relaciones.

Figura 57. Páginas 25 – 28 Libro “Programación intermedios”



2.3.6. Ventajas

UML permite una serie de ventajas al ser utilizadas. Algunas ventajas son que UML:

- Permite mayor énfasis en la especificación.
- Permite la realización de una verificación y validación del modelo.
- Permite la automatización de procesos determinados y la generación de código a partir de los modelos.
- Es independiente de la plataforma de desarrollo.

2.3.7. Vistas

UML utiliza cinco vistas para representar a los sistemas. Las cinco vistas son:

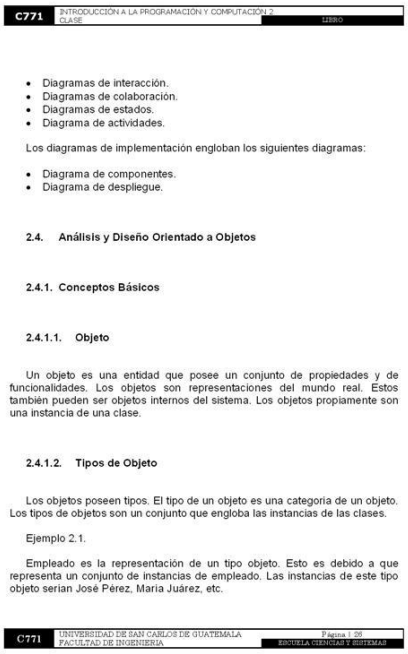
- Vista de usuario: representa lo que l usuario percibe del sistema.
- Vista estructural: representa el estado estático del sistema.
- Vista de comportamiento: esta vista se conoce también como vista de proceso. Esta vista representa el estado dinámico del sistema.
- Vista de implementación: representa la distribución de los componentes del sistema y su relación con el hardware.
- Vista de ambiente: representa los elementos físicos del sistema (hardware).

2.3.8. Diagramas

UML utiliza una serie de diagramas que son utilizados en cada una de las vistas. Los diagramas más importantes son: el diagrama de clases y el diagrama de secuencia.

El diagrama de clases representa la estructura del sistema. El diagrama de secuencia presenta el comportamiento entre actores y objetos del sistema.

Los diagramas de comportamiento engloban los siguientes diagramas:



- Diagramas de interacción.
- Diagramas de colaboración.
- Diagramas de estados.
- Diagrama de actividades.

Los diagramas de implementación engloban los siguientes diagramas:

- Diagrama de componentes.
- Diagrama de despliegue.

2.4. Análisis y Diseño Orientado a Objetos

2.4.1. Conceptos Básicos

2.4.1.1. Objeto

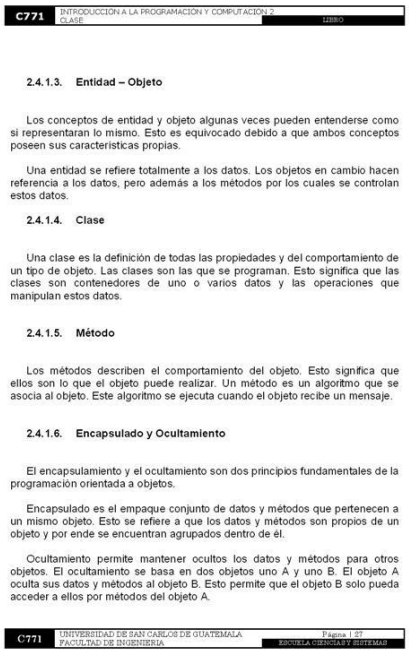
Un objeto es una entidad que posee un conjunto de propiedades y de funcionalidades. Los objetos son representaciones del mundo real. Estos también pueden ser objetos internos del sistema. Los objetos propiamente son una instancia de una clase.

2.4.1.2. Tipos de Objeto

Los objetos poseen tipos. El tipo de un objeto es una categoría de un objeto. Los tipos de objetos son un conjunto que engloba las instancias de las clases.

Ejemplo 2.1.

Empleado es la representación de un tipo objeto. Esto es debido a que representa un conjunto de instancias de empleado. Las instancias de este tipo objeto serían José Pérez, María Juárez, etc.



2.4.1.3. Entidad – Objeto

Los conceptos de entidad y objeto algunas veces pueden entenderse como si representaran lo mismo. Esto es equivocado debido a que ambos conceptos poseen sus características propias.

Una entidad se refiere totalmente a los datos. Los objetos en cambio hacen referencia a los datos, pero además a los métodos por los cuales se controlan estos datos.

2.4.1.4. Clase

Una clase es la definición de todas las propiedades y del comportamiento de un tipo de objeto. Las clases son las que se programan. Esto significa que las clases son contenedores de uno o varios datos y las operaciones que manipulan estos datos.

2.4.1.5. Método

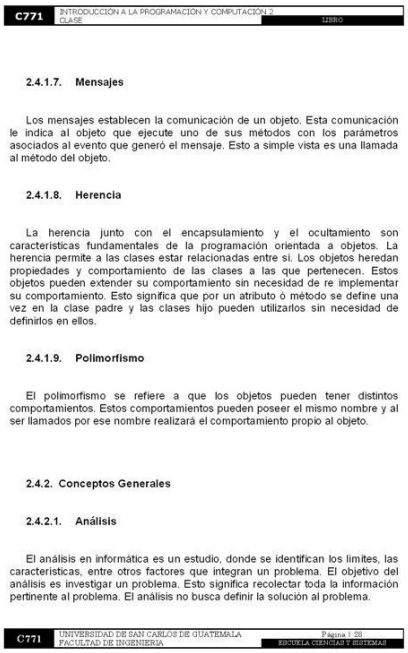
Los métodos describen el comportamiento del objeto. Esto significa que ellos son lo que el objeto puede realizar. Un método es un algoritmo que se asocia al objeto. Este algoritmo se ejecuta cuando el objeto recibe un mensaje.

2.4.1.6. Encapsulado y Ocultamiento

El encapsulamiento y el ocultamiento son dos principios fundamentales de la programación orientada a objetos.

Encapsulado es el empaque conjunto de datos y métodos que pertenecen a un mismo objeto. Esto se refiere a que los datos y métodos son propios de un objeto y por ende se encuentran agrupados dentro de él.

Ocultamiento permite mantener ocultos los datos y métodos para otros objetos. El ocultamiento se basa en dos objetos uno A y uno B. El objeto A oculta sus datos y métodos al objeto B. Esto permite que el objeto B solo pueda acceder a ellos por métodos del objeto A.



2.4.1.7. Mensajes

Los mensajes establecen la comunicación de un objeto. Esta comunicación le indica al objeto que ejecute uno de sus métodos con los parámetros asociados al evento que generó el mensaje. Esto a simple vista es una llamada al método del objeto.

2.4.1.8. Herencia

La herencia junto con el encapsulamiento y el ocultamiento son características fundamentales de la programación orientada a objetos. La herencia permite a las clases estar relacionadas entre sí. Los objetos heredan propiedades y comportamiento de las clases a las que pertenecen. Estos objetos pueden extender su comportamiento sin necesidad de re implementar su comportamiento. Esto significa que por un atributo ó método se define una vez en la clase padre y las clases hijo pueden utilizarlos sin necesidad de definirlos en ellos.

2.4.1.9. Polimorfismo

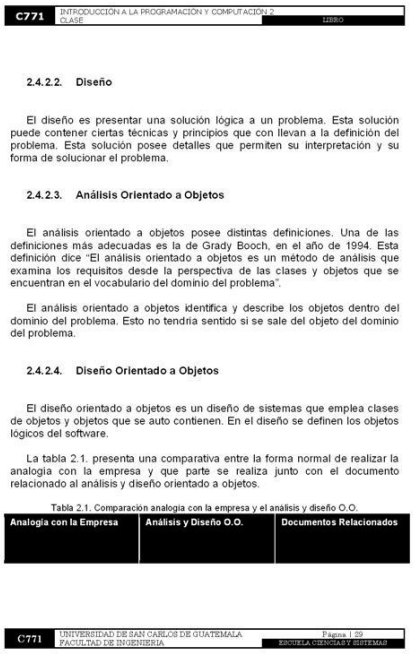
El polimorfismo se refiere a que los objetos pueden tener distintos comportamientos. Estos comportamientos pueden poseer el mismo nombre y al ser llamados por ese nombre realizará el comportamiento propio al objeto.

2.4.2. Conceptos Generales

2.4.2.1. Análisis

El análisis en informática es un estudio, donde se identifican los límites, las características, entre otros factores que integran un problema. El objetivo del análisis es investigar un problema. Esto significa recolectar toda la información pertinente al problema. El análisis no busca definir la solución al problema.

Figura 58. Páginas 29 – 32 Libro “Programación intermedios”



2.4.2.2. Diseño

El diseño es presentar una solución lógica a un problema. Esta solución puede contener ciertas técnicas y principios que con llevan a la definición del problema. Esta solución posee detalles que permiten su interpretación y su forma de solucionar el problema.

2.4.2.3. Análisis Orientado a Objetos

El análisis orientado a objetos posee distintas definiciones. Una de las definiciones más adecuadas es la de Grady Booch, en el año de 1994. Esta definición dice “El análisis orientado a objetos es un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema”.

El análisis orientado a objetos identifica y describe los objetos dentro del dominio del problema. Esto no tendría sentido si se sale del objeto del dominio del problema.

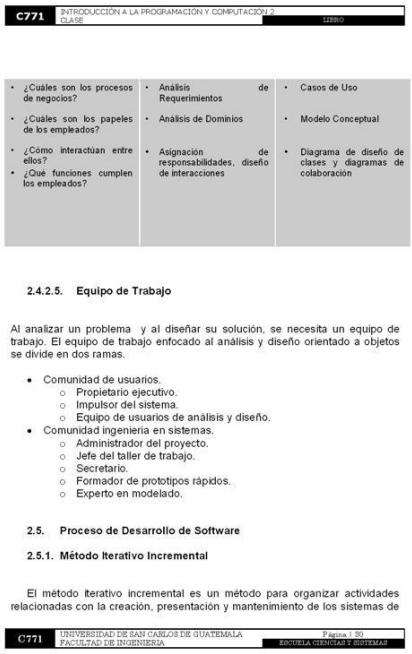
2.4.2.4. Diseño Orientado a Objetos

El diseño orientado a objetos es un diseño de sistemas que emplea clases de objetos y objetos que se auto contienen. En el diseño se definen los objetos lógicos del software.

La tabla 2.1. presenta una comparativa entre la forma normal de realizar la analogía con la empresa y que parte se realiza junto con el documento relacionado al análisis y diseño orientado a objetos.

Tabla 2.1. Comparación analogía con la empresa y el análisis y diseño O.O.

Analogía con la Empresa	Análisis y Diseño O.O.	Documentos Relacionados



• ¿Cuáles son los procesos de negocios? • ¿Cuáles son los papeles de los empleados? • ¿Cómo interactúan entre ellos? • ¿Qué funciones cumplen los empleados?	• Análisis de Requerimientos • Análisis de Dominios • Asignación de responsabilidades, diseño de interacciones	• Casos de Uso • Modelo Conceptual • Diagrama de diseño de clases y diagramas de colaboración

2.4.2.5. Equipo de Trabajo

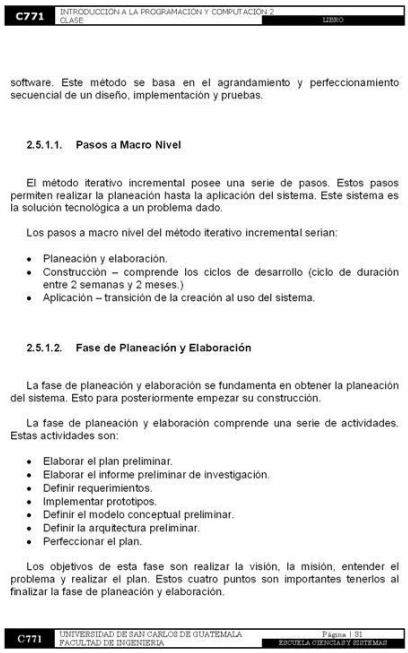
Al analizar un problema y al diseñar su solución, se necesita un equipo de trabajo. El equipo de trabajo enfocado al análisis y diseño orientado a objetos se divide en dos ramas.

- Comunidad de usuarios.
  - Propietario ejecutivo.
  - Impulsor del sistema.
  - Equipo de usuarios de análisis y diseño.
- Comunidad ingeniería en sistemas.
  - Administrador del proyecto.
  - Jefe del taller de trabajo.
  - Secretario.
  - Formador de prototipos rápidos.
  - Experto en modelado.

2.5. Proceso de Desarrollo de Software

2.5.1. Método Iterativo Incremental

El método iterativo incremental es un método para organizar actividades relacionadas con la creación, presentación y mantenimiento de los sistemas de



software. Este método se basa en el agrandamiento y perfeccionamiento secuencial de un diseño, implementación y pruebas.

2.5.1.1. Pasos a Macro Nivel

El método iterativo incremental posee una serie de pasos. Estos pasos permiten realizar la planeación hasta la aplicación del sistema. Este sistema es la solución tecnológica a un problema dado.

Los pasos a macro nivel del método iterativo incremental serían:

- Planeación y elaboración.
- Construcción – comprende los ciclos de desarrollo (ciclo de duración entre 2 semanas y 2 meses.)
- Aplicación – transición de la creación al uso del sistema.

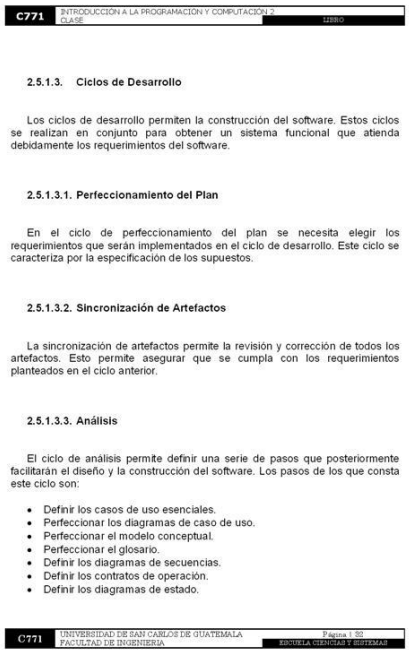
2.5.1.2. Fase de Planeación y Elaboración

La fase de planeación y elaboración se fundamenta en obtener la planeación del sistema. Esto para posteriormente empezar su construcción.

La fase de planeación y elaboración comprende una serie de actividades. Estas actividades son:

- Elaborar el plan preliminar.
- Elaborar el informe preliminar de investigación.
- Definir requerimientos.
- Implementar prototipos.
- Definir el modelo conceptual preliminar.
- Definir la arquitectura preliminar.
- Perfeccionar el plan.

Los objetivos de esta fase son realizar la visión, la misión, entender el problema y realizar el plan. Estos cuatro puntos son importantes tenerlos al finalizar la fase de planeación y elaboración.



2.5.1.3. Ciclos de Desarrollo

Los ciclos de desarrollo permiten la construcción del software. Estos ciclos se realizan en conjunto para obtener un sistema funcional que atienda debidamente los requerimientos del software.

2.5.1.3.1. Perfeccionamiento del Plan

En el ciclo de perfeccionamiento del plan se necesita elegir los requerimientos que serán implementados en el ciclo de desarrollo. Este ciclo se caracteriza por la especificación de los supuestos.

2.5.1.3.2. Sincronización de Artefactos

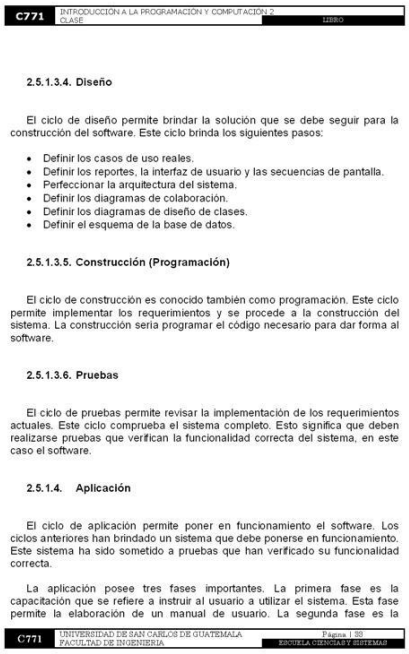
La sincronización de artefactos permite la revisión y corrección de todos los artefactos. Esto permite asegurar que se cumpla con los requerimientos planteados en el ciclo anterior.

2.5.1.3.3. Análisis

El ciclo de análisis permite definir una serie de pasos que posteriormente facilitarán el diseño y la construcción del software. Los pasos de los que consta este ciclo son:

- Definir los casos de uso esenciales.
- Perfeccionar los diagramas de caso de uso.
- Perfeccionar el modelo conceptual.
- Perfeccionar el glosario.
- Definir los diagramas de secuencias.
- Definir los contratos de operación.
- Definir los diagramas de estado.

Figura 59. Páginas 33 – 36 Libro “Programación intermedios”



2.5.1.3.4. Diseño

El ciclo de diseño permite brindar la solución que se debe seguir para la construcción del software. Este ciclo brinda los siguientes pasos:

- Definir los casos de uso reales.
- Definir los reportes, la interfaz de usuario y las secuencias de pantalla.
- Perfeccionar la arquitectura del sistema.
- Definir los diagramas de colaboración.
- Definir los diagramas de diseño de clases.
- Definir el esquema de la base de datos.

2.5.1.3.5. Construcción (Programación)

El ciclo de construcción es conocido también como programación. Este ciclo permite implementar los requerimientos y se procede a la construcción del sistema. La construcción sería programar el código necesario para dar forma al software.

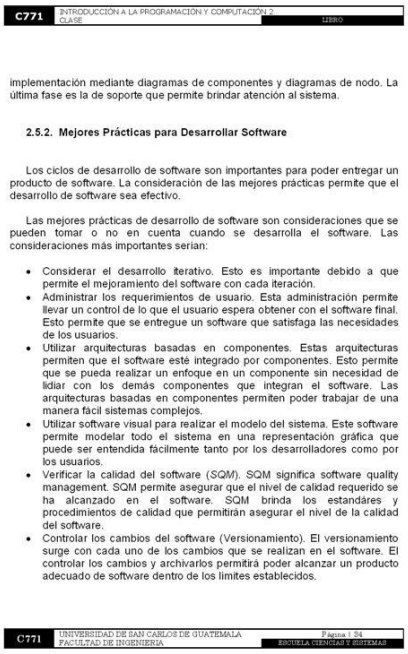
2.5.1.3.6. Pruebas

El ciclo de pruebas permite revisar la implementación de los requerimientos actuales. Este ciclo comprueba el sistema completo. Esto significa que deben realizarse pruebas que verifican la funcionalidad correcta del sistema, en este caso el software.

2.5.1.4. Aplicación

El ciclo de aplicación permite poner en funcionamiento el software. Los ciclos anteriores han brindado un sistema que debe ponerse en funcionamiento. Este sistema ha sido sometido a pruebas que han verificado su funcionalidad correcta.

La aplicación posee tres fases importantes. La primera fase es la capacitación que se refiere a instruir al usuario a utilizar el sistema. Esta fase permite la elaboración de un manual de usuario. La segunda fase es la



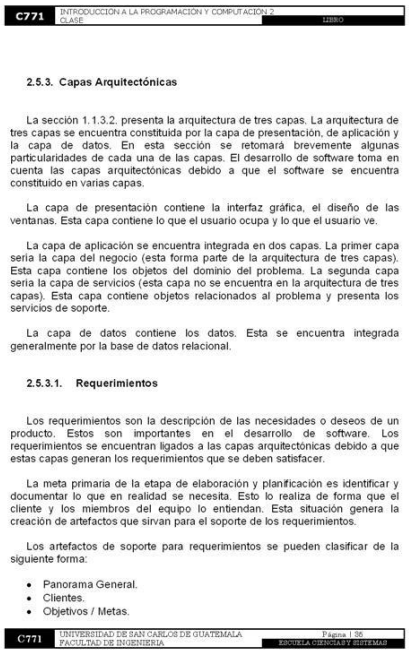
implementación mediante diagramas de componentes y diagramas de nodo. La última fase es la de soporte que permite brindar atención al sistema.

2.5.2. Mejores Prácticas para Desarrollar Software

Los ciclos de desarrollo de software son importantes para poder entregar un producto de software. La consideración de las mejores prácticas permite que el desarrollo de software sea efectivo.

Las mejores prácticas de desarrollo de software son consideraciones que se pueden tomar o no en cuenta cuando se desarrolla el software. Las consideraciones más importantes serían:

- Considerar el desarrollo iterativo. Esto es importante debido a que permite el mejoramiento del software con cada iteración.
- Administrar los requerimientos de usuario. Esta administración permite llevar un control de lo que el usuario espera obtener con el software final. Esto permite que se entregue un software que satisfaga las necesidades de los usuarios.
- Utilizar arquitecturas basadas en componentes. Estas arquitecturas permiten que el software esté integrado por componentes. Esto permite que se pueda realizar un enfoque en un componente sin necesidad de lidiar con los demás componentes que integran el software. Las arquitecturas basadas en componentes permiten poder trabajar de una manera fácil sistemas complejos.
- Utilizar software visual para realizar el modelo del sistema. Este software permite modelar todo el sistema en una representación gráfica que puede ser entendida fácilmente tanto por los desarrolladores como por los usuarios.
- Verificar la calidad del software (*SQM*). SQM significa software quality management. SQM permite asegurar que el nivel de calidad requerido se ha alcanzado en el software. SQM brinda los estandáres y procedimientos de calidad que permitirán asegurar el nivel de la calidad del software.
- Controlar los cambios del software (Versionamiento). El versionamiento surge con cada uno de los cambios que se realizan en el software. El controlar los cambios y archivarlos permitirá poder alcanzar un producto adecuado de software dentro de los límites establecidos.



2.5.3. Capas Arquitectónicas

La sección 1.1.3.2. presenta la arquitectura de tres capas. La arquitectura de tres capas se encuentra constituida por la capa de presentación, de aplicación y la capa de datos. En esta sección se retomará brevemente algunas particularidades de cada una de las capas. El desarrollo de software toma en cuenta las capas arquitectónicas debido a que el software se encuentra constituido en varias capas.

La capa de presentación contiene la interfaz gráfica, el diseño de las ventanas. Esta capa contiene lo que el usuario ocupa y lo que el usuario ve.

La capa de aplicación se encuentra integrada en dos capas. La primer capa sería la capa del negocio (esta forma parte de la arquitectura de tres capas). Esta capa contiene los objetos del dominio del problema. La segunda capa sería la capa de servicios (esta capa no se encuentra en la arquitectura de tres capas). Esta capa contiene objetos relacionados al problema y presenta los servicios de soporte.

La capa de datos contiene los datos. Esta se encuentra integrada generalmente por la base de datos relacional.

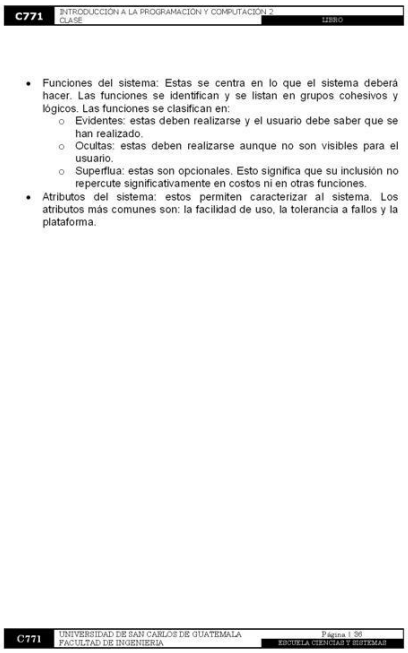
2.5.3.1. Requerimientos

Los requerimientos son la descripción de las necesidades o deseos de un producto. Estos son importantes en el desarrollo de software. Los requerimientos se encuentran ligados a las capas arquitectónicas debido a que estas capas generan los requerimientos que se deben satisfacer.

La meta primaria de la etapa de elaboración y planificación es identificar y documentar lo que en realidad se necesita. Esto lo realiza de forma que el cliente y los miembros del equipo lo entiendan. Esta situación genera la creación de artefactos que sirvan para el soporte de los requerimientos.

Los artefactos de soporte para requerimientos se pueden clasificar de la siguiente forma:

- Panorama General.
- Clientes.
- Objetivos / Metas.



- Funciones del sistema: Estas se centra en lo que el sistema deberá hacer. Las funciones se identifican y se listan en grupos cohesivos y lógicos. Las funciones se clasifican en:
  - Evidentes: estas deben realizarse y el usuario debe saber que se han realizado.
  - Ocultas: estas deben realizarse aunque no son visibles para el usuario.
  - Superflua: estas son opcionales. Esto significa que su inclusión no repercute significativamente en costos ni en otras funciones.
- Atributos del sistema: estos permiten caracterizar al sistema. Los atributos más comunes son: la facilidad de uso, la tolerancia a fallos y la plataforma.

Figura 60. Páginas 37 – 40 Libro “Programación intermedios”

3. ETAPA DE ANÁLISIS DEL CICLO DE CONSTRUCCIÓN

3.1. Casos de Uso

3.1.1. Introducción

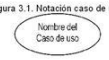
3.1.1.1. Definición

Los casos de uso son un documento narrativo que describe la secuencia de eventos de un actor que utiliza un sistema para completar un proceso. El actor es un agente externo. Los casos de uso siempre ejemplifican las historias que narran.

3.1.1.2. Notación

La notación de los casos de uso, es decir su representación gráfica se realiza mediante una elipse. Esta elipse representa al caso de uso y en su interior contiene el nombre del caso de uso. El nombre de los casos de uso debe iniciar siempre con un verbo. La notación de un caso de uso se representa en la figura 3.1.

Figura 3.1. Notación caso de uso



Ejemplo 3.1.

Se necesita realizar un caso de uso que represente la compra de productos en determinada empresa. El nombre del caso de uso que represente la acción sería comprar producto. La figura 3.2. representa el caso de uso comprar producto.

Figura 3.2. Caso de uso comprar producto

3.1.1.3. Modalidades Generales de Casos de Uso

Los casos de uso poseen dos modalidades generales. Estas modalidades representan a los casos de uso. Las modalidades permiten apreciar a mayor o menor nivel de detalle los casos de uso. Estas modalidades son:

- Casos de uso de alto nivel.
- Casos de uso expandidos.

3.1.1.3.1. CDU de Alto Nivel

Los casos de uso de alto nivel describen un proceso muy brevemente. Esto significa que su nivel de detalle es considerablemente bajo. Estos casos de uso se utilizan durante el examen inicial de requerimientos y del proyecto. El objetivo de estos casos de uso es entender rápidamente el grado de complejidad y funcionalidad del sistema.

La representación de este tipo de casos de uso contiene:

- Caso de uso: identificador y el nombre del caso de uso.
- Actores: son los agentes que intervienen en el caso de uso.
- Tipo: el tipo de caso de uso, en este caso si es primario o secundario.
- Descripción: permite narrar la historia concerniente al caso de uso.

Ejemplo 3.2.

A partir del caso de uso representado en la figura 3.2., realizar el caso de uso de alto nivel. El caso de uso representa la compra de un producto. El identificador que se le asigna al caso de uso es CDU-001, CDU indica que es un caso de uso y el 001 representa el número de caso de uso a tratar. En esta acción intervienen dos actores: el cajero y el cliente. Este tipo de caso de uso es del tipo primario debido a que su acción es importante y no depende de ningún otro caso de uso. La tabla 3.1. contiene la solución del caso de uso comprar producto.

Tabla 3.1. CDU de alto nivel, comprar producto

Tipo	Descripción
Caso de uso	CDU-001 Comprar Producto
Actores	Cajero, Cliente
Tipo	Primario
Descripción	El cliente llega a la caja registradora con los artículos y cobra el importe. Al terminar la operación el cliente se marcha con los productos.

3.1.1.3.2. CDU Expandidos

Los casos de uso expandidos poseen un nivel de detalle mayor a los casos de uso de alto nivel. Los casos de uso expandidos generalmente son más útiles para alcanzar un mayor conocimiento de los procesos y requerimientos del sistema. Los casos de uso expandidos normalmente utilizan un estilo coloquial para describir la interacción entre los actores y el sistema.

Los casos de uso se estructuran de la siguiente forma:

- Caso de uso: identificador y nombre del caso de uso.
- Actores: los agentes que intervienen en el caso de uso.
- Propósito: el objetivo del caso de uso.
- Resumen: narra las historia de las acciones contenidas en el caso de uso.
- Tipo: si es primario o secundario.
- Referencia cruzada: significa si el caso de uso emplea otro caso de uso y por consiguiente lo referencia.
- Sección principal
    - Curso normal de eventos: representa los eventos que ocurren en un escenario ideal, es decir donde no existe ningún problema.
        - Evento 1
        - Evento 2
        - ...

    - Curso alterno de eventos: representa los eventos que pueden ocurrir generados por algún problema que altere el escenario ideal.
        - Evento 1
        - Evento 2
        - ...

Ejemplo 3.3.

A partir del ejemplo 3.2, realizar el caso de uso expandido para el caso de uso comprar producto. Este ejemplo toma como base el caso de uso de alto nivel. A diferencia del CDU alto nivel añade el propósito que conlleva el caso de uso en este caso es capturar una venta y su pago en efectivo. En la sección principal se realizará toda la secuencia de eventos ideal para poder realizar la compra de un producto (curso normal de eventos). En el curso alterno de eventos se detallará cada uno de los eventos que pueden surgir y que de alguna forma alterarán el curso normal de eventos. Las referencias cruzadas se incluyen para hacer referencia a otros casos de uso. Este ejemplo hace referencia a otros casos de uso que no se incluyen en el ejemplo y que son representados solamente por su identificador. La tabla 3.2. brinda la solución al caso de uso expandido comprar productos en efectivo.

Tabla 3.2. CDU expandido, comprar productos en efectivo

Tipo	Descripción
Caso de uso	CDU-001 Comprar productos en efectivo
Actores	Cliente (iniciador), Cajero
Propósito	Capturar una venta y su pago en efectivo
Resumen	El cliente llega a la caja registradora con los productos que comprará. El cajero registra los artículos y recibe un pago en efectivo. Al terminar la operación el cliente se marcha con los productos.
Tipo	Primario, Esencial
Refs. Cruzadas	CDU-002
[Sección Principal]	

Figura 61. Páginas 41 – 44 Libro “Programación intermedios”

Curso Normal de Eventos	1. El cliente llega a caja con los productos que desea comprar. 2. El cajero registra el identificador de cada producto. Si hay varios productos de una misma categoria el cajero también puede introducir la cantidad. 3. El sistema determina el precio del producto e incorpora a la transacción actual la información correspondiente. 4. Al terminar de introducir los productos, el cajero indica al sistema que terminó la captura del producto. 5. El sistema calcula y presenta el total de la venta. 6. El cajero indica el total al cliente. 7. El cliente efectúa el pago en efectivo posiblemente mayor al total de la venta. 8. El cajero registra la cantidad de efectivo recibida. 9. El sistema muestra al cliente la diferencia, genera una factura. 10. El cajero deposita el dinero recibido y extrae el cambio del pago. El cajero entrega al cliente el cambio y la factura impresa. 11. El sistema registra la venta concluida. 12. El cliente se marcha con los artículos comprados.
Cursos Alternos	• Línea 2: Se introduce un indicador inválido. Indicar error. • Línea 7: El cliente no tiene suficiente dinero. Cancelar la transacción de venta.

3.1.1.4. Tipos de Casos de Uso

Los casos de uso poseen diversos tipos. Las tablas de los CDU de alto nivel y expandidos hacen referencia al tipo de los casos de uso. Los casos de uso pueden ser de dos tipos el caso de prioridad (caso 1) y el físico (caso 2).

- Caso 1.
  - Primario: el CDU primario representa procesos comunes e importantes.
  - Secundario: el CDU secundario representa los procesos menores o raros.
  - Opcional: el CDU opcional representa procesos que podrían no abordarse.
- Caso 2.

- Esencial: el CDU esencial es conocido también como abstracto. Este CDU se considera sin compromiso con el diseño. Este es utilizado en la fase de análisis.
- Real: el CDU real es conocido también como concreto. Este CDU se encuentra comprometido con el diseño. Este se usa en la fase de diseño.

3.1.2. Actores

3.1.2.1. Definición

Los actores son las entidades externas al sistema que de alguna forma participan en la historia de los casos de uso. Los actores pueden representar papeles de seres humanos, otros sistemas informáticos, dispositivos electrónicos, etc. Esto debe entenderse que de ninguna forma representan al sistema que se está construyendo debido a que los actores son agentes externos. Esto indica que si ellos formaran parte del sistema serían internos y esto contradice a la definición.

3.1.2.2. Notación

Los actores en el diagrama de casos de uso se representan por medio de un *stickman*. El nombre del actor se coloca debajo de las piernas del *stickman*. El nombre debe ser lo más representativo al actor. La figura 3.3. presenta la notación de un actor.

Figura 3.3. Notación de un actor

3.1.3. Identificación de Casos de Uso

Los casos de uso pueden ser identificados de diversas formas. Estas formas varían desde una lluvia de ideas hasta por una revisión de la documentación sobre la especificación de requerimientos. Los métodos más comunes para identificar casos de uso son el basado en actores y el basado en eventos.

3.1.3.1. Método Basado en Actores

El método basado en actores empieza identificando los actores que se relacionan con el sistema o con la empresa. El siguiente paso es que para cada actor se identifiquen los procesos que inician o en los que participan. Este método permite de esta forma identificar los casos de uso y a la vez permite identificar fácilmente los actores.

3.1.3.2. Método Basado en Eventos

El método basado en eventos empieza identificando los eventos externos a los que un sistema debe responder. El siguiente paso es relacionar los eventos con los actores y con los casos de uso.

3.1.4. Diagramas de Casos de Uso

3.1.4.1. Consideraciones

Los diagramas de caso de uso son una forma fácil de representar los sistemas. Las consideraciones que se deben seguir para poder realizar un diagrama de casos de uso son las siguientes:

- El caso de uso es representado por una elipse.
- Los actores son representados por *stickman*.
- Los casos de uso se colocan dentro de un cuadro denominado frontera.
- La frontera establece el límite de las cosas que son externas o internas a ellas.
- Los actores van afuera de la frontera.

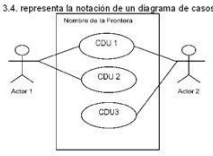
- Actores y casos se comunican por una línea.

3.1.4.2. Diagrama

El diseño del diagrama de casos de uso debe contemplar las consideraciones de la sección 3.1.4.1.

La frontera es el cuadro en cuyo interior se colocan los casos de uso. El nombre de la frontera se coloca en la esquina superior izquierda. Los actores deben ser colocados fuera de la frontera. Los actores se relacionan con los casos de uso por medio de una línea.

La figura 3.4. representa la notación de un diagrama de casos de uso



Ejemplos 3.4.

A partir del ejemplo 3.3. se realizará el diagrama de casos de uso para que contenga comprar producto. Los actores que intervienen se encuentran descritos en la tabla 3.2. En este caso se han identificado el actor cliente y el actor cajero. La frontera se determina con el nombre de punto de venta. La figura 3.5. muestra el diagrama de casos de uso correspondiente.

Figura 3.5. Diagrama CDU comprar producto

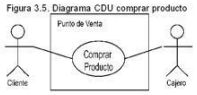


Figura 62. Páginas 45 – 48 Libro “Programación intermedios”

3.1.5. Sistemas y sus Fronteras

Las fronteras representan el ámbito donde se encuentra comprendido el sistema. Los diagramas de casos de uso utilizan las fronteras para comprender los casos de uso. Estas fronteras se encuentran representadas por un cuadrado.

3.1.5.1. Fronteras Ordinarias

Las fronteras ordinarias representan los ámbitos comunes donde se representan los sistemas. Algunas fronteras ordinarias serian las siguientes:

- El hardware y software de un dispositivo de computo.
- El departamento de una organización.
- La organización entera.

3.1.5.2. Utilización de CDU

Los casos de uso se aplican en las fases de planeación y elaboración. Estos también se aplican en la fase de construcción de la metodologia iterativa/incremental.

En la fase de planeación y elaboración los casos de uso se utilizan cuando ya se tenga la lista de funciones del sistema. Posterior a esto se debe:

- Definir la frontera de este y luego identificar los actores y los CDU.
- Escribir todos los CDU en formato de alto nivel y clasificarlos en primario, secundarios u opcionales
- Dibujar el diagrama de casos de uso.
- Relacionar los CDU y dar ejemplos de las relaciones en el diagrama.
- Escribir en formato esencial expandido los CDU más importantes, influyentes y más riesgosos a fin de entender y estimar mejor la naturaleza y las dimensiones del problema.

En la fase de análisis se deben elaborar CDU expandidos esenciales de los CDU incluidos en el ciclo de construcción. En la fase de diseño se deben elaborar los CDU expandidos reales de los CDU incluidos en este ciclo.

3.1.5.3. Ventajas

Los casos de uso brindan diversas ventajas. Algunas de estas ventajas son:

- Los CDU aseguran el desarrollo del sistema correcto.
- Los CDU documentan respuestas funcionales de caja negra.
- Los CDU ayudan a gestionar la complejidad del proyecto.
- Los CDU proporcionan el fundamento de los mensajes.
- Los CDU ofrecen una buena base para la verificación y validación.
- Los CDU son un modo objetivo para el seguimiento del proyecto.
- Los CDU sirven de base para especificar respuestas de aplicaciones de tiempo real.

3.1.6. Relaciones en un Diagrama de Casos de Uso

Los casos de uso no se encuentran aislados. Estos muchas veces se relacionan con otros casos de uso dentro de la frontera del sistema. Esto debido a que un sistema se encuentra formado por uno o más casos de uso. Las relaciones que utilizan los casos de uso son: *communicates*, *include*, *extend*, generalización.

3.1.6.1. Communicates

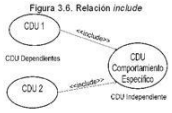
La relación communicates es una relación entre un actor y un caso de uso. Esta relación denota la participación del actor en el caso de uso. Esta relación indica que el actor interviene en el caso de uso donde aparece esta relación.

3.1.6.2. Include

La relación de *include* es una relación directa entre dos casos de uso. Esta relación implica que el comportamiento del caso de uso incluido es insertado dentro del comportamiento del caso de uso al que se incluye. Los casos de uso incluidos son obligatorios y siempre son requeridos para ejecutar correctamente el caso de uso en el que se incluyen.

La representación gráfica de la relación *include* se realiza por medio de una flecha sólida desde el caso de uso dependiente hasta el caso de uso independiente. La palabra *include* se coloca entre los simbolos << y >>. Esto permite indicar el tipo de relación que se genera en los casos de uso. La figura 3.6. representa gráficamente la relación *include*.

Figura 3.6. Relación *include*

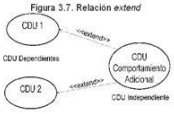


3.1.6.3. Extend

La relación *extend* también es una relación entre dos casos de uso. Esta relación indica que el comportamiento de un caso de uso puede ser extendido por el comportamiento de otro caso de uso. El caso de uso que se extiende es significativamente independiente del caso de uso extendido.

La representación gráfica de la relación *extend* se realiza por medio de una flecha punteada desde el caso de uso independiente hasta el caso de uso dependiente. La palabra *extend* se coloca entre los simbolos << y >>. Esto permite indicar el tipo de relación que se genera en los casos de uso. La figura 3.7. representa gráficamente la relación *extend*.

Figura 3.7. Relación *extend*



3.1.6.4. Generalización

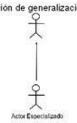
Un caso de uso extiende otro caso de uso añadiendo acciones al caso de uso general. Esto permite indicar ciertos casos de uso son más generales que otros casos de uso. La generalización indica que un caso de uso es una variante de otro caso de uso. El caso de uso especificado puede variar cualquier aspecto del caso de uso base.

La representación gráfica de la relación de generalización se representa en la figura 3.8. La generalización es una flecha sólida que parte del caso de uso especifico hacia el caso de uso general.

Figura 3.8. Relación de generalización

Los actores también pueden encontrarse generalizados. Esto significa que un actor puede especializarse y generalizarse. La figura 3.9. representa la generalización de un actor. La flecha sólida parte del actor especializado hacia el actor general.

Figura 3.9. Relación de generalizacion entre actores



Ejemplo 3.5.

A partir del ejemplo 3.6. se extenderá la funcionalidad del caso de uso comprar producto. Esto se realizará añadiendo un caso de uso devolver producto. Asimismo se generalizará el caso de uso comprar producto y de esta forma se crearán dos casos de uso especificos que son comprar en efectivo y

Figura 63. Páginas 49 – 52 Libro “Programación intermedios”

comprar con tarjeta de crédito. El caso de uso identificar usuario se vuelve importante debido a que en el sistema se debe verificar el usuario que realiza la transacción. Los actores que intervienen en este caso de uso son el cajero y el cliente. La figura 3.10. representa el caso de uso comprar producto extendido.

Figura 3.10. Caso de uso comprar producto extendido

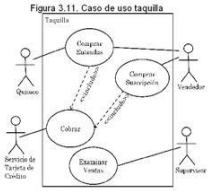
Ejemplo 3.6.

Una taquilla es el lugar en el cual se venden entradas para poder acceder a un evento público. La compra de una entrada se puede realizar por medio de un quiosco. Los vendedores pueden comprar entradas y ellos se suscriben a un servicio para poder realizar su compra. El cobro de las entradas se puede realizar por medio de un servicio de tarjetas de crédito. El proceso de las ventas es examinado por un supervisor de ventas.

Los actores de este enunciado son: el supervisor, el vendedor y el quiosco. El quiosco puede ser catalogado como otro sistema que permite pedidos de un cliente. El cliente no pertenece al sistema debido a que no se encuentra conectado directamente con la taquilla.

Los casos de uso son: comprar entradas (esto se realiza por medio del quiosco o del vendedor), examinar ventas totales (esto se realiza por el supervisor). Los cobros de las entradas se realizan por medio del servicio de tarjeta de crédito. Estos casos de uso son los más elementales del sistema de taquilla. Esto quiere decir que pueden existir más casos de uso que pueden aumentar el nivel de detalle del diagrama. La figura 3.11. muestra el diagrama de casos de uso del sistema de taquilla.

Figura 3.11. Caso de uso taquilla



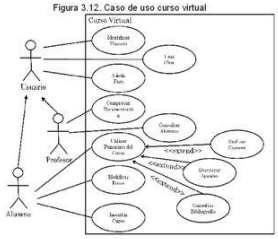
Ejemplo 3.7.

Un curso virtual es una opción factible para que los estudiantes puedan acceder de forma remota hacia los beneficios de un curso. Esto permite que un estudiante pueda elegir entre una serie de opciones. Las opciones principales serian el de identificación, uso de chat, foro, ver el listado de los documentos para el curso y de los alumnos, modificar sus datos, inscribirse, realizar funciones del curso como realizar exámenes, descargar los apuntes, consultar la bibliografía. Este sistema es utilizado tanto por el alumno como el profesor.

Los actores identificados serán una extensión del actor usuario. Este usuario puede ser profesor y alumno.

Los casos de uso identificados para el usuario son identificar usuario, usar chat, asistir al foro. Los casos de uso identificados para el profesor son comprobar documentación y consultar alumnos. Los casos de uso identificados para el alumno son modificar datos, inscribirse en el curso, realizar funciones del curso, el cual se extenderá a realizar examen, descargar apuntes y consultar bibliografía. La figura 3.12. muestra el caso de uso del sistema del curso virtual.

Figura 3.12. Caso de uso curso virtual



3.2. Diagramas de Estructura Estática

3.2.1. Diagrama de Clases

El diagrama de clases es un diagrama estático que describe la estructura del sistema denotando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases se utilizan durante el proceso de análisis y diseño del ciclo de desarrollo de software.

3.2.1.1. Clase

Las clases son el componente principal de un diagrama de clases. Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. La clase dentro de un diagrama de clases se representa con un rectángulo con tres divisiones internas.

3.2.1.2. Atributo

Las clases contienen atributos. Un atributo es una característica propia de una clase. Los atributos poseen su propia sintaxis para poder ser escritos e interpretados como tal. La sintaxis de un atributo es el siguiente:

- Visibilidad NombreAtributo : tipo = [valor inicial]

El tipo se refiere al tipo de valor que posee el atributo que puede variar desde un tipo primitivo entero hasta un tipo clase. La visibilidad de los atributos se maneja considerando tres tipos. La visibilidad puede ser:

- Protegida: Esta se representa con el símbolo #.
- Pública: Esta se representa con el símbolo +.
- Privada: Esta se representa con el símbolo -.

3.2.1.3. Operación y Método

Las clases contienen operaciones que permiten realizar ciertas funciones propias de la clase. Una operación describe el comportamiento de los objetos de una clase. La sintaxis de una operación es la siguiente:

- Visibilidad NombreOperacion (lista de parámetros).

Las operaciones dentro de las clases son conocidas como métodos. Los métodos son la implementación de una operación. La sintaxis de un método es la siguiente:

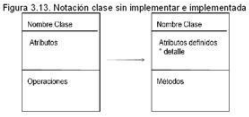
- Visibilidad NombreMetodo (lista parámetro con tipo) : [valor de retorno]

3.2.1.4. Notación de una Clase

Las clases se representan gráficamente con un rectángulo con dos líneas horizontales. Esto hace que el rectángulo se divida en tres áreas. La parte superior contiene el nombre de la clase. La parte del medio contiene los atributos. La parte inferior contiene las operaciones ó métodos. La figura 3.13. muestra la forma de ver una clase antes de implementarla y posterior a implementarla.

Figura 64. Páginas 53 – 56 Libro “Programación intermedios”

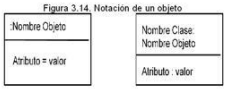
Figura 3.13. Notación clase sin implementar e implementada



3.2.1.5. Objeto (Instancia)

Las clases son estáticas y representan un conjunto de objetos. Un objeto es entonces una instancia de una clase. La sección 2.4.1.1. detalla mejor lo que un objeto es. Esta sección presenta la notación dentro del diagrama de clases de un objeto. La notación de un objeto es semejante a la de una clase. La diferencia radica en que la parte superior del rectángulo el nombre del objeto inicia con dos puntos (:). Este nombre también puede contener el nombre de la clase antes de los dos puntos. Otra diferencia es que en la sección de los atributos, éstos poseen valores iniciales. La figura 3.14. presenta ambas formas de notación de un objeto.

Figura 3.14. Notación de un objeto



Ejemplo 3.8.

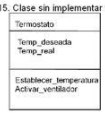
Un termostato es un componente que permite abrir o cerrar un circuito eléctrico en función de la temperatura para encender un ventilador. Se desea realizar una clase que presente una clase sin implementar, implementada y el objeto correspondiente.

Lo primero que se debe de realizar es establecer el nombre de la clase, en este caso será termostato. Lo siguiente es establecer los atributos. Los atributos identificados son temperatura deseada y temperatura real. Las

operaciones que debe realizar la clase son establecer la temperatura y activar el ventilador.

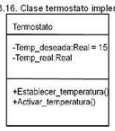
La figura 3.15. muestra la clase sin implementar termostato. Esta clase posee bajo nivel de detalle.

Figura 3.15. Clase sin implementar termostato



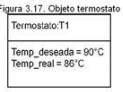
La figura 3.16. presenta la clase implementada termostato. Esta clase contiene la visibilidad propia de los atributos y los métodos.

Figura 3.16. Clase termostato implementada



La figura 3.17. muestra la clase termostato instanciada. Este objeto se llama T1 y sus atributos poseen valores que lo establecen como objeto.

Figura 3.17. Objeto termostato



3.2.1.6. Relaciones entre Clases

Las clases se relacionan entre si de distinta forma. Esto sucede debido a que las clases no se encuentran aisladas. Ellas se encuentran integradas como un conjunto. Las relaciones que existen entre las clases son:

- Asociación binaria
- Asociación n-aria
- Composición
- Generalización
- Dependencia
- Relación de refinamiento

Las relaciones entre las clases presentan conceptos que deben entenderse antes de explicar a detalle cada uno de los tipos de relaciones. Los conceptos son el rol y la multiplicidad.

El rol en las relaciones se identifica como un nombre a los finales de la línea. Esto quiere decir que una relación se representa como una línea. El rol describe la semántica de la relación en el sentido indicado.

La multiplicidad indica la cardinalidad de la relación. Existen cuatro formas de representar la multiplicidad:

- 1: significa es uno.
- *: significa es muchos
- 1..*: significa uno o más
- 0..*: significa cero o más.

3.2.1.6.1. Asociación (Binaria, N-aria)

La asociación binaria es una relación que se representa como una línea sólida que une dos clases. La asociación binaria no exige dependencia existencial ni encapsulamiento.

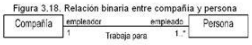
La asociación N-aria es una relación entre tres o más clases. Esta relación se representa gráficamente mediante un diamante del cual salen líneas de asociación a todas las clases que comprenden esta asociación.

Ejemplo 3.9.

Una compañía está compuesta por muchos empleados. Realizar la representación mediante una relación binaria.

Las relaciones ocurren entre clases. Las clases se representan como un rectángulo. Los conceptos identificados son compañía y persona. El concepto persona es más general que indicar empleado. Esto es debido a que empleado es una especificación de persona. Estas dos son clases. Una compañía es empleadora de uno o más empleados. La figura 3.18. muestra como se representa la relación binaria entre una compañía y una persona.

Figura 3.18. Relación binaria entre compañía y persona



Ejemplo 3.10.

Un cliente de un banco posee un servicio de cuentas. Esto quiere decir que un cliente puede tener muchas cuentas dentro del banco.

La figura 3.19. muestra la relación binaria entre cuenta y cliente. La relación que existe es que un cliente es el titular de una cuenta.

Figura 3.19. Relación binaria entre

Ejemplo 3.11.

Un jugador que pertenece a un equipo cada año realiza los apuntes necesarios donde guarda sus records alcanzados cada año en una temporada. Realizar su representación mediante una relación N-aria.

Las clases identificadas son equipo, jugador y año. Estas tres clases se relacionan cada una mediante una relación N-aria. Varios equipos pueden tener varios jugadores. Varios equipos y jugadores pueden participar en varias temporadas. La multiplicidad elegida para todas las clases es muchas. La figura 3.20. muestra la relación N-aria entre un equipo, un jugador y un año.

Figura 65. Páginas 57 – 60 Libro “Programación intermedios”

Figura 3.20. Relación N-aria equipo, jugador, año

Ejemplo 3.12.

Un avión es un concepto, el cual puede ser piloteado por un piloto y en el cual viajan pasajeros.

La figura 3.21. muestra la relación N-aria entre un avión, un piloto y un pasajero.

Figura 3.21. Relación N-aria entre avión, pasajero y piloto

3.2.1.6.2. Composición

La composición es una relación que a diferencia de la asociación exige una dependencia existencial. Esto es debido a que existe en este tipo de relación una pertenencia fuerte entre las clases. Los objetos contenidos no se encuentran compartidos. La composición se denota dibujando un rombo relleno del lado de la clase que contiene a la otra clase.

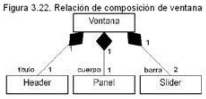
Ejemplo 3.13.

Una ventana de un programa se encuentra integrada por una cabecera, el panel y el slider. Esta ventana permite realizar diversas operaciones en las aplicaciones. Se desea realizar una relación que represente la relación de composición.

Las clases identificadas son ventana, header, panel y slider. La clase ventana se denota que se encuentra compuesta por la clase header, panel y

slider. Una ventana se encuentra compuesta por un header, un panel y 2 slider. La figura 3.22. muestra la relación de composición de ventana.

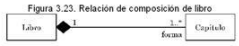
Figura 3.22. Relación de composición de ventana



Ejemplo 3.14.

Un libro es un concepto el cual se encuentra integrado por capítulos. Los capítulos permiten formar el contenido del libro. Estos pueden estar constituidos de diversas formas. La figura 3.23. muestra esta relación.

Figura 3.23. Relación de composición de libro



3.2.1.6.3. Generalización

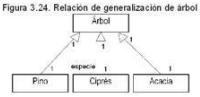
La generalización es una relación que denota herencia entre clases. Esto significa que una clase puede heredar conducta de otra clase. La relación de generalización se representa por un triángulo sin rellenar en el lado de la superclase. La superclase es la clase que hereda comportamiento y conducta a las subclases.

Ejemplo 3.15.

Un árbol es un una planta, que posee un tronco el cual se ramifica a una altura determinada. Existe variedad tipos de arboles. Los tipos de arboles pueden ser pinos, cipreses y acacias. Realizar una representación de una relación de generalización de un árbol.

Las clases identificadas son árbol, pino, ciprés y acacia. Árbol es la superclase que hereda a las subclases pino, ciprés y acacia. Estas tres últimas clases son especies de árboles. La figura 3.24. presenta la relación de generalización de la clase árbol.

Figura 3.24. Relación de generalización de árbol

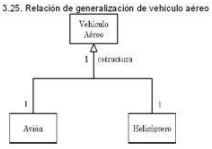


Ejemplo 3.16.

Los vehículos aéreos permiten a las personas el poder desplazarse por el aire. Estos vehículos poseen diversidad de tipos dependiendo de su estructura. Los vehículos aéreos dependiendo su estructura se pueden dividir en aviones y helicópteros.

La figura 3.25. presenta la relación de generalización de la clase vehículo aéreo.

Figura 3.25. Relación de generalización de vehículo aéreo



3.2.1.6.4. Dependencia

La dependencia es una relación semántica entre dos elementos del modelo de clases. La dependencia indica cambiar el elemento independiente que pude requerir cambios en los elementos dependientes. La dependencia se representa con una línea punteada direccional, indicando el sentido de la dependencia.

3.2.1.7. Notas

Las notas permiten en los diagramas de clases añadir información detallada de una clase. Esta información es adicional a los atributos y métodos de la

clase. Las notas se añaden para facilitar la comprensión de ciertas clases que podrían resultar confusos su entendimiento. La ventaja de las notas es que se pueden anclar a los elementos *UML* para indicar su pertenencia al objeto.

3.2.1.8. Estrategia para Identificar Conceptos

Las estrategias para identificar conceptos permiten de una manera fácil identificar cada uno de los conceptos de los cuales se pueden formar las clases. Al identificar los conceptos se puede distinguir si estos conforman una clase o no.

3.2.1.8.1. Directrices para Construir el Modelo Conceptual

Un modelo conceptual es una descripción del ámbito de un problema real. Esto significa que un modelo conceptual explica los conceptos que son significativos del problema. El modelo conceptual se diferencia del diagrama de clases en que este último contiene una descripción de las entidades del software en lugar de los conceptos del mundo real. Los conceptos son útiles para poder construir las clases.

Las directrices que permiten identificar conceptos son las siguientes:

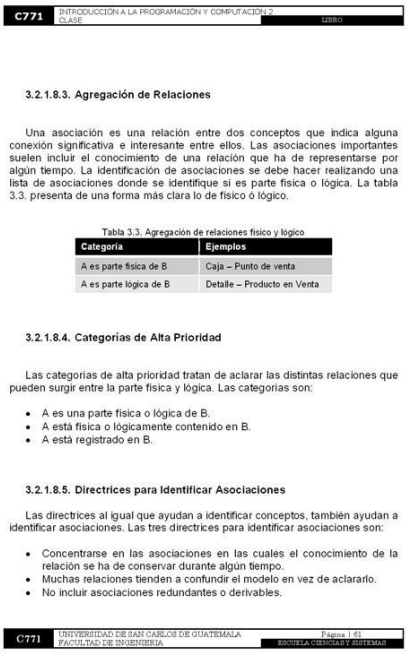
- Listar los conceptos idóneos usando la lista de categorías de conceptos.
- Dibujar los conceptos en un modelo conceptual.
- Incorporar las relaciones necesarias.
- Agregar los atributos.

3.2.1.8.2. Asignación de Nombres y Modelado de los Conceptos

La importancia de asignar nombres adecuados para distinguir los conceptos es importante. Esto puede prevenir la confusión del modelo. A continuación se listan unas consideraciones al respecto:

- Utilizar nombres existentes en el territorio.
- Excluir características irrelevantes.
- No agregar cosas que no existen.

Figura 66. Páginas 61 – 64 Libro “Programación intermedios”



3.2.1.8.3. Agregación de Relaciones

Una asociación es una relación entre dos conceptos que indica alguna conexión significativa e interesante entre ellos. Las asociaciones importantes suelen incluir el conocimiento de una relación que ha de representarse por algún tiempo. La identificación de asociaciones se debe hacer realizando una lista de asociaciones donde se identifique si es parte física o lógica. La tabla 3.3. presenta de una forma más clara lo de físico ó lógico.

Tabla 3.3. Agregación de relaciones físico y lógico

Categoría	Ejemplos
A es parte física de B	Caja – Punto de venta
A es parte lógica de B	Detalle – Producto en Venta

3.2.1.8.4. Categorías de Alta Prioridad

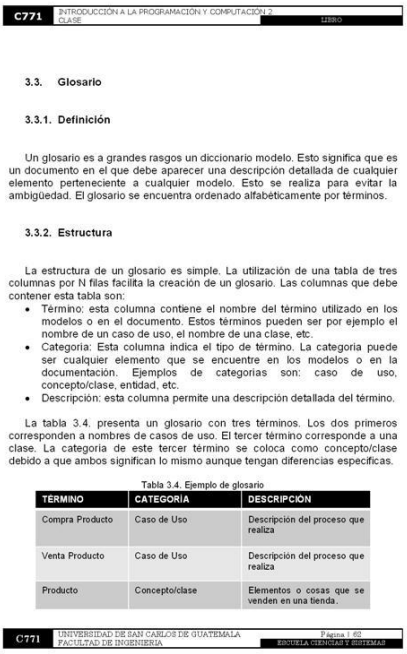
Las categorías de alta prioridad tratan de aclarar las distintas relaciones que pueden surgir entre la parte física y lógica. Las categorías son:

- A es una parte física o lógica de B.
- A está física o lógicamente contenido en B.
- A está registrado en B.

3.2.1.8.5. Directrices para Identificar Asociaciones

Las directrices al igual que ayudan a identificar conceptos, también ayudan a identificar asociaciones. Las tres directrices para identificar asociaciones son:

- Concentrarse en las asociaciones en las cuales el conocimiento de la relación se ha de conservar durante algún tiempo.
- Muchas relaciones tienden a confundir el modelo en vez de aclararlo.
- No incluir asociaciones redundantes o derivables.



3.3. Glosario

3.3.1. Definición

Un glosario es a grandes rasgos un diccionario modelo. Esto significa que es un documento en el que debe aparecer una descripción detallada de cualquier elemento perteneciente a cualquier modelo. Esto se realiza para evitar la ambigüedad. El glosario se encuentra ordenado alfabéticamente por términos.

3.3.2. Estructura

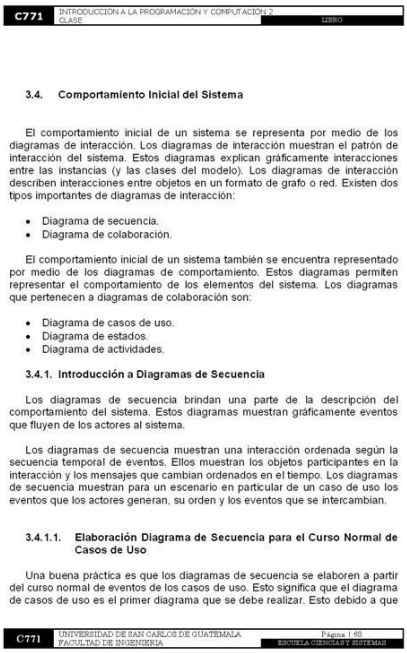
La estructura de un glosario es simple. La utilización de una tabla de tres columnas por N filas facilita la creación de un glosario. Las columnas que debe contener esta tabla son:

- Término: esta columna contiene el nombre del término utilizado en los modelos o en el documento. Estos términos pueden ser por ejemplo el nombre de un caso de uso, el nombre de una clase, etc.
- Categoría: Esta columna indica el tipo de término. La categoría puede ser cualquier elemento que se encuentre en los modelos o en la documentación. Ejemplos de categorías son: caso de uso, concepto/clase, entidad, etc.
- Descripción: esta columna permite una descripción detallada del término.

La tabla 3.4. presenta un glosario con tres términos. Los dos primeros corresponden a nombres de casos de uso. El tercer término corresponde a una clase. La categoría de este tercer término se coloca como concepto/clase debido a que ambos significan lo mismo aunque tengan diferencias específicas.

Tabla 3.4. Ejemplo de glosario

TÉRMINO	CATEGORÍA	DESCRIPCIÓN
Compra Producto	Caso de Uso	Descripción del proceso que realiza
Venta Producto	Caso de Uso	Descripción del proceso que realiza
Producto	Concepto/clase	Elementos o cosas que se venden en una tienda.



3.4. Comportamiento Inicial del Sistema

El comportamiento inicial de un sistema se representa por medio de los diagramas de interacción. Los diagramas de interacción muestran el patrón de interacción del sistema. Estos diagramas explican gráficamente interacciones entre las instancias (y las clases del modelo). Los diagramas de interacción describen interacciones entre objetos en un formato de grafo o red. Existen dos tipos importantes de diagramas de interacción:

- Diagrama de secuencia.
- Diagrama de colaboración.

El comportamiento inicial de un sistema también se encuentra representado por medio de los diagramas de comportamiento. Estos diagramas permiten representar el comportamiento de los elementos del sistema. Los diagramas que pertenecen a diagramas de colaboración son:

- Diagrama de casos de uso.
- Diagrama de estados.
- Diagrama de actividades.

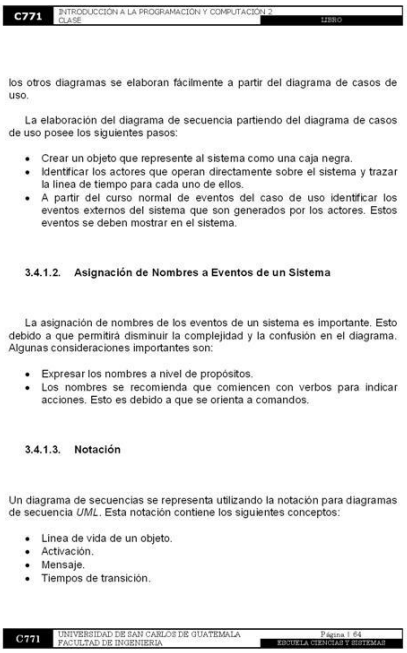
3.4.1. Introducción a Diagramas de Secuencia

Los diagramas de secuencia brindan una parte de la descripción del comportamiento del sistema. Estos diagramas muestran gráficamente eventos que fluyen de los actores al sistema.

Los diagramas de secuencia muestran una interacción ordenada según la secuencia temporal de eventos. Ellos muestran los objetos participantes en la interacción y los mensajes que cambian ordenados en el tiempo. Los diagramas de secuencia muestran para un escenario en particular de un caso de uso los eventos que los actores generan, su orden y los eventos que se intercambian.

3.4.1.1. Elaboración Diagrama de Secuencia para el Curso Normal de Casos de Uso

Una buena práctica es que los diagramas de secuencia se elaboren a partir del curso normal de eventos de los casos de uso. Esto significa que el diagrama de casos de uso es el primer diagrama que se debe realizar. Esto debido a que



los otros diagramas se elaboran fácilmente a partir del diagrama de casos de uso.

La elaboración del diagrama de secuencia partiendo del diagrama de casos de uso posee los siguientes pasos:

- Crear un objeto que represente al sistema como una caja negra.
- Identificar los actores que operan directamente sobre el sistema y trazar la línea de tiempo para cada uno de ellos.
- A partir del curso normal de eventos del caso de uso identificar los eventos externos del sistema que son generados por los actores. Estos eventos se deben mostrar en el sistema.

3.4.1.2. Asignación de Nombres a Eventos de un Sistema

La asignación de nombres de los eventos de un sistema es importante. Esto debido a que permitirá disminuir la complejidad y la confusión en el diagrama. Algunas consideraciones importantes son:

- Expresar los nombres a nivel de propósitos.
- Los nombres se recomienda que comiencen con verbos para indicar acciones. Esto es debido a que se orienta a comandos.

3.4.1.3. Notación

Un diagrama de secuencias se representa utilizando la notación para diagramas de secuencia *UML*. Esta notación contiene los siguientes conceptos:

- Línea de vida de un objeto.
- Activación.
- Mensaje.
- Tiempos de transición.

Figura 67. Páginas 65 – 68 Libro “Programación intermedios”

3.4.1.3.1. Línea de Vida de un Objeto

La linea de vida de un objeto representa para el objeto su vida durante la interacción. La linea de vida se representa como una línea vertical punteada con un rectángulo de encabezado y con rectángulos a través de la línea principal que denotan la ejecución de métodos (activación). El rectángulo de encabezado contiene el nombre del objeto y de su clase en un formato nombreObjeto:nombreClase. Al finalizar el tiempo de vida de un objeto se coloca un circulo con una x dentro del él. Esto representa que se ha acabado la vida de ese objeto.

3.4.1.3.2. Activación

La activación es el periodo de tiempo durante el cual, el objeto se encuentra desarrollando alguna operación. La activación es denotada con un rectángulo delgado sobre la linea de vida del objeto.

3.4.1.3.3. Mensaje

El concepto de mensaje se ha establecido en la sección 2.1.1.7. Los mensajes en el diagrama de secuencias se denotan por medio de una linea sólida dirigida. Esta dirección parte desde el objeto que emite el mensaje hacia el objeto que lo ejecuta.

3.4.1.3.4. Tiempo de Transición

Los tiempos de transición ocurren entre los tiempos de salida y llegada de los mensajes. Esto indica que existen objetos concurrentes o demoras en la recepción de mensajes. Esto ocasiona que se le añadan nombres a estos tiempos.

Ejemplo 3.17.

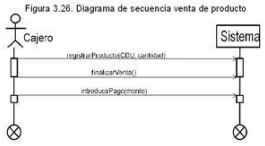
Se desea realizar un diagrama de secuencias que represente el curso normal de eventos de un diagrama de caso de uso. El curso normal de eventos se encuentra dado en la Tabla 3.5.

Tabla 3.5. Curso normal de eventos caso de uso venta producto

Curso normal de eventos
1. En todos los productos el cajero registra el código universal de producto (CUD) y la cantidad. 2. Al terminar de capturar los productos, el cajero indica al punto de venta que la venta concluyó. 3. El cajero indica el total al cliente y este da un pago. 4. El cajero registra el importe ofrecido en efectivo.

El caso de uso describe la situación en el que el cajero interactúa con el sistema de ventas. Esto lo realiza por medio del ingreso de los productos y del tipo de pago. La figura 3.26. muestra el diagrama de secuencia asociado a la tabla 3.5. El cajero es el actor principal que interactúa con el punto de venta. El cajero realiza tres operaciones: registrar el producto, finalizar la venta e introducir el pago. Estas operaciones son vistas como mensajes. La activación se realizará en dos momentos. El primero cuando el cajero registra el producto y finaliza la venta. El segundo cuando el cajero introduce el pago.

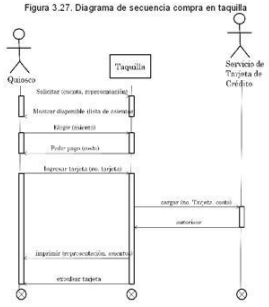
Figura 3.26. Diagrama de secuencia venta de producto



Ejemplo 3.18.

A partir del diagrama de casos de uso de la figura 3.11. del ejemplo 3.6. realizar el diagrama de secuencia que determine la interacción entre los actores y los casos de uso. Los actores que intervienen dentro de esta interacción son el quiosco, la taquilla y el servicio de tarjeta de crédito. El actor supervisor es omitido en este diagrama de secuencia debido a que participa en otro momento con el sistema. Esto quiere decir que el diagrama de secuencia permitirá visualizar el cobro hecho en la taquilla y el examen de las ventas se omite debido a que se debe realizar otro diagrama de secuencias para el mismo. La figura 3.27. muestra este diagrama de secuencia.

Figura 3.27. Diagrama de secuencia compra en taquilla



Ejemplo 3.19.

Realizar un diagrama de secuencias de un proceso concerniente a la solicitud de compra de un producto. La secuencia es la siguiente:

1. Un cliente solicita una cotización hacia un proveedor.
2. El proveedor le envía la cotización al cliente.
3. El cliente decide enviar una contraoferta hacia su proveedor.
4. El proveedor le envía al cliente la oferta.
5. El cliente entonces decide enviar la orden de compra.
6. El proveedor decide enviar la confirmación de recibir la orden de compra y posteriormente despachara la orden de compra al cliente.
7. El cliente acepta el despacho y el proveedor le envía la factura.
8. El cliente entonces realiza el pago de la orden de compra y el proceso finaliza.

El diagrama de secuencia que permite representar el proceso de compra de productos de un cliente con su proveedor se encuentra en la figura 3.28.

Figura 3.28. Diagrama de secuencia compra productos

3.4.2. Introducción a Diagramas de Estado

Los diagramas de estados permiten mostrar la secuencia de estados por la que pasa un caso de uso, un objeto o todo el sistema.

Los diagramas de estado indican que eventos provocan el cambio de estado y cuáles son las respuestas y acciones que genera. Estos diagramas contienen todos los mensajes que un objeto puede enviar o recibir. El escenario representa un camino dentro del diagrama.

Los diagramas de estado al igual que los de secuencia se describen mediante la notación *UML*. Estos diagramas de estado contienen conceptos importantes tales como el estado, el evento, envio de mensajes, transición simple, transición interna.

Figura 68. Páginas 69 – 72 Libro “Programación intermedios”

3.4.2.1. Estado

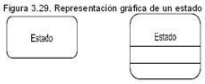
Un estado identifica un periodo del objeto en el cual el objeto espera una operación, tiene un estado característico o puede recibir cierto tipo de estímulo.

La representación gráfica de un estado es un rectángulo con los bordes redondeados. El estado puede contener tres segmentos:

- Nombre del estado.
- Valor característico de los atributos del objeto en ese estado.
- Segmento para las acciones que se realizan al entrar, salir o estar en un estado.

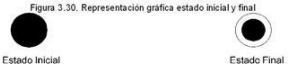
La figura 3.29. muestra la representación gráfica de un estado sin segmentos y un estado con segmentos.

Figura 3.29. Representación gráfica de un estado



Los estados poseen dos tipos especiales. El estado inicial que indica el inicio del diagrama de estados. Este se representa con un circulo lleno. El estado final que indica el fin del diagrama de estados. Este se representa con un circulo lleno dentro de un circulo vacio. La figura 3.30. representa el estado inicial y el estado final.

Figura 3.30. Representación gráfica estado inicial y final



Los estados pueden ser independientes y dependientes. Los estados independientes se caracterizan en que un objeto reacciona de distinta forma ante un mensaje dependiendo de su estado. Los estados dependientes se caracterizan debido a que un objeto reacciona siempre igual a un mensaje dependiendo de su estado. Algunos tipos y clases comúnmente dependientes del estado son:

- Sistemas.
- Ventanas.
- Coordinadores de aplicaciones.
- Controladores.
- Transacciones.
- Dispositivos.
- Mutadores.

3.4.2.2. Eventos

Un evento es una acción que puede originar la transición de un estado a otro estado de un objeto. El nombre de un evento posee alcance dentro del paquete en el cual se define, no es local a la clase que lo nombre. Los eventos pueden tomar cualquier representación de las siguientes:

- Una condición que toma el valor de verdadero o falso.
- La recepción de una señal de otro objeto.
- La recepción de un mensaje.
- El paso de cierto periodo de tiempo, después de entrar al estado o de cierta hora y fecha en particular.

Los eventos pueden ser externos, internos o temporales. Los eventos externos se deben a un factor situado fuera de la frontera del sistema. Los eventos internos tienen lugar cuando se invoca un mensaje o señal que partió de otro objeto interno. Los eventos temporales se deben a la ocurrencia de una fecha u hora específica o al transcurso del tiempo.

3.4.2.3. Envío de Mensajes

El envio de mensajes dentro de un diagrama de estados se realiza por medio de una linea punteada dirigida al diagrama de estados del objeto receptor del mensaje.

3.4.2.4. Transición Simple

La transición simple es una relación entre dos estados. Esta relación indica que un objeto en el primer estado puede entrar al segundo estado. Este objeto

puede ejecutar entonces ciertas operaciones cuando cierto evento sucede y si ciertas condiciones son satisfechas.

La transición simple se representa por medio de una linea sólida entre dos estados. Esta linea puede contener texto con el siguiente formato: *event-signature* [*guard-condition*] '/' *action-expression* '^' *send-clause*. Este formato representa ciertos conceptos que representan la transición. Los significados de estos son:

- *event-signature*: esta representa la descripción del evento que origina la transición.
- *guard-condition*: esta representa las condiciones adicionales al evento que se necesitan para que ocurra la transición.
- *action-expression*: esta representa el mensaje al objeto o a otro objeto que es ejecutado como resultado de la transacción y el cambio de estado.
- *send-clause*: esta representa acciones adicionales que se ejecutan con el cambio de estado.

3.4.2.5. Transición Interna

La transición interna ocurre cuando se permanece en el mismo estado en lugar de involucrar dos estados distintos. Esto significa que una transición interna representa un evento que no causa un cambio de estado. La transición interna se denota como una cadena adicional en el compartimiento de acciones del estado.

La figura 3.31. muestra la representación gráfica de un diagrama de estados incluyendo un dos estados uno simple y otro compuesto, eventos, transiciones y estados.

Figura 3.31. Representación gráfica de un diagrama de estados

Ejemplo 3.20.

Un teléfono es un dispositivo de telecomunicación. Este aparato permite comunicar a los seres humanos. El proceso de realizar una llamada empieza por medio de descolgar el auricular, marcar, realizar la llamada y conversar. El marcado se realiza añadiendo un dígito de uno en uno. Realizar el diagrama de estados correspondiente.

El diagrama de estados al igual que el diagrama de secuencias se realiza a partir de un caso de uso. El enunciado del ejemplo permite comprender el curso normal de eventos de realizar una llamada por teléfono. Esto permite realizar el diagrama de eventos. Los estados identificados son 4. Estos son: descolgar, marcado parcial, llamada en progreso y conversación en progreso. Estos estados describen el proceso de realizar una llamada. El estado descolgar cambia al momento en que se presiona un dígito esto permite alcanzar el estado marcado parcial. Este estado se vuelve mediante una transición interna que es presionar dígito al mismo estado. Al momento en que se alcanza el número deseado se procede al estado llamada en progreso. Este estado espera mientras se atiende la llamada y se alcanza el estado de conversación en progreso. Este estado es el último del proceso realizar llamada.

La figura 3.32. muestra el diagrama de estados para el proceso de realizar una llamada.

Figura 3.32. Diagrama de estado realizar llamada telefónica

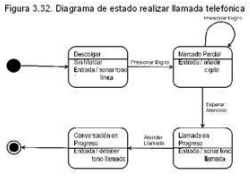


Figura 69. Páginas 73 – 76 Libro “Programación intermedios”

Ejemplo 3.21.

El proceso de compra de compra de un producto se caracteriza en que existe un punto de venta donde se introducen los productos. Estos productos luego de que se han listado todos se procede a realizar su pago. Este pago puede ser en efectivo, cheque ó tarjeta de crédito. Estos dos últimos tipos de pago se encuentran sujetos a una autorización de pago. A partir de esto realizar el diagrama de estados correspondiente.

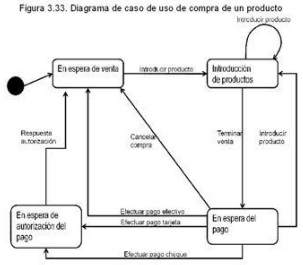
Los estados que se identifican son: en espera de venta, introducción de productos, en espera del pago y en espera de autorización del pago. Este diagrama de casos de uso no posee estado final debido a que el proceso siempre está en espera de la siguiente transacción.

La espera de venta se interrumpe al introducir un producto. Esto conlleva al estado de introducción de productos donde se pueden introducir más productos. Este estado cambia al estado de en espera del pago cuando se termina la venta. En el estado en espera del pago se encuentran tres situaciones.

La primera es volver al estado introducción de productos si se introduce un nuevo producto. La segunda es pasar al estado de en espera de venta si el pago es efectuado en efectivo. La tercera es pasar al estado en espera de autorización del pago si el pago se efectúa con tarjeta de crédito o con un cheque. Este estado es liberado cuando se obtiene una respuesta a la autorización. La respuesta aunque sea aprobada o reprobada llega al estado inicial.

La figura 3.33. muestra el diagrama de estados correspondiente al proceso de compra de un producto.

Figura 3.33. Diagrama de caso de uso de compra de un producto



Ejemplo 3.22.

El proceso de la compra de las entradas de una taquilla permite que el vendedor pueda añadir a su suscripción las entradas. Esto quiere decir que cuando solicita una entrada disponible esta puede quedar bloqueada si se autoriza que se aparte. El caso de que se arrepienta de comprarla puede desbloquear el apartar la entrada. La entrada apartada también puede ser puesta en disponible si el tiempo para que se comprara expira. La entrada debe siempre venderse y esto permite regresar a la disponibilidad.

Los estados disponibles son tres. Estos son disponible, bloqueado y vendido.

La figura 3.34. muestra el diagrama de estados correspondiente al proceso de venta de una entrada.

Figura 3.34. Diagrama de caso de uso de compra de un producto

4. ETAPA DE DISEÑO DEL CICLO DE CONSTRUCCIÓN

4.1. Reportes

Los reportes son la forma en que se les presenta la información de un sistema a los usuarios. Los reportes pueden ser diseñados por medio de hojas de diseño de reporte en pantalla o en papel. El diccionario de datos es fuente de datos para los reportes. A los usuarios se les presentan modelos o prototipos de reportes antes de terminar el diseño del reporte.

4.2. Interfaz de Usuario

La interfaz de usuario es el medio mediante el cual el usuario se comunica con la computadora.

La interfaz de usuario es utilizada para desempeñar diversas funciones. Las principales de la interfaz de usuario son:

- Servir como herramienta de desarrollo de aplicaciones.
- Establecer comunicación con otros sistemas.
- Establecer la configuración de la interfaz y el entorno.
- Permitir el intercambio de datos entre las aplicaciones.

4.2.1. Interfaz Gráfica de Usuario

Las siglas en inglés de la interfaz gráfica de usuario son *GUI*. *GUI* significa *graphics user interfaces*.

Las interfaces gráficas de usuario son artefactos que permiten mediante el uso y representación del lenguaje visual, una interacción amigable con el sistema. Las *GUI* se caracterizan por ser gráficas y por permitir una interacción rápida e intuitiva.

Figura 70. Páginas 77 – 80 Libro “Programación intermedios”

4.3. Secuencia de Pantallas

Las secuencias de pantallas no son más que la presentación de las pantallas del software propio del sistema. Esto significa que se presenta de manera ordenada y lógica desde la pantalla inicial hasta la última pantalla en la que se puede acceder. Esto es refiriéndose al software.

La secuencia de pantallas posee una estructura. Esta estructura permite de manera ordenada presentar a los usuarios las pantallas. La estructura permite indicar que se puede realizar una tabla de tres columnas. Esto donde la primera columna será el número de la pantalla. La segunda columna será la pantalla. La tercer columna serán las observaciones para la pantalla.

Ejemplo 4.1.

Un cajero automático es una máquina que permite a los usuarios poder extraer dinero de ella. El cajero automático posee un menú que permite al usuario utilizarlo con facilidad. El menú del cajero automático posee un menú inicial donde se muestran las operaciones que el usuario puede realizar. Las opciones a su vez poseen operaciones específicas. Realice la secuencia de pantallas donde se muestre el menú principal y un menú cualquier operación del menú principal.

Las operaciones del menú principal se establecen como operaciones básicas, solicitudes, servicios, compras. Las operaciones básicas contienen transferencias y extracciones. Las solicitudes son claves y solicitudes. Los servicios son especiales y link de pagos. Las compras serían automáticas y depósitos.

La operación del menú a realizar será el link de pagos. Este contiene operaciones tales como pago de impuestos y servicios, consultas, baja en empresas adheridas, pago con depósito.

Aplicando la estructura brindada en esta sección se realizará la tabla correspondiente a la secuencia de pantallas. La secuencia de pantallas se puede realizar con una imagen o por medio de texto. La imagen detalla mejor la pantalla a mostrar. El texto debe estar claramente definido y mostrado para que se entienda la pantalla. Este es utilizado en etapas iniciales del diseño debido a que no se posee ninguna pantalla hecha. La tabla 4.1. muestra la secuencia de pantallas para el cajero automático.

Tabla 4.1. Secuencia de pantallas del cajero ATM

Número de Pantalla	Diseño	Observaciones
1	Seleccione el tipo de operación que desea efectuar < Transferencias Extracciones > Solicitudes < Claves Solicitudes > Servicios < Especiales Link Pagos > Compra Cajero < Automático Depósitos >	Menú inicial de ATM
2	Seleccione el tipo de operación Pago Impuestos y Servicios > Consultas > Baja empresas adheridas > Pago C/Deposito de <Factura o Boleta Rentas Global>	Menú Principal Links de Pagos

4.4. Diagrama de Clases

Los diagramas de clases fueron descritos a detalle en la sección 3.2. La sección 4.4. se centra en conceptos más complejos de los diagramas de clases. Esta sección presenta conceptos aplicados al diseño del software utilizando diagramas de clases de uso.

La sección 3.2. se centró en describir los diagramas, presentando su notación y sus relaciones. La sección 4.4. presenta la forma en que los conceptos de la sección 3.2. se pueden utilizar para crear el diagrama de clases.

Los diagramas de clases muestran las definiciones de las entidades del software. Este diagrama contiene:

- Clases y asociaciones.
- Atributos con su visibilidad, tipo y valores iniciales cuando aplican.
- Interfaces con sus operaciones y constantes.
- Navegabilidad.
- Dependencia.

4.4.1. Dependencias

El diagrama de clases depende directamente de dos diagramas. El modelo conceptual y los diagramas de interacción.

El modelo conceptual permite que a partir de este modelo crear los detalles de la definición de clases.

Los diagramas de interacción permiten que a partir de estos, identificar las clases de software y los métodos de dichas clases.

4.4.2. Relaciones de Dependencia

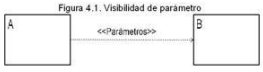
Las relaciones de dependencia se dan cuando una clase conoce a otra clase por un medio que no es a través de un atributo. En estas relaciones un objeto debe conocer a otro para poder llamar a sus métodos. Esto permite decir que el primer objeto tiene visibilidad sobre el segundo objeto.

Las relaciones de dependencia permiten determinar tres tipos de visibilidad.

- Parámetro
- Local
- Global

La visibilidad de parámetro ocurre al momento en que a un método de una clase le es pasado como parámetro un objeto de otra clase. Esto permite indicar que la clase primera posee visibilidad de parámetro sobre la segunda clase. La figura 4.1. presenta la visibilidad de parámetro.

Figura 4.1. Visibilidad de parámetro



La visibilidad local de dependencia ocurre al momento en que en un método de una clase se define una variable local que pertenece como objeto a otra clase. Esto permite indicar que la clase primera tiene visibilidad local sobre la clase segunda. La figura 4.2. muestra la visibilidad local.

Figura 4.2. Visibilidad local

La visibilidad global de dependencia ocurre cuando existe una variable global en el sistema, por ejemplo una instancia de una clase X, y un método de una clase Y realiza una llamada a un método de la clase X. Esto permite indicar que la clase X tiene visibilidad global de la clase A. La figura 4.3 muestra esta dependencia.

Figura 4.3. Visibilidad global

4.4.3. Clase Controladora

La clase controladora es un caso especial de visibilidad global. Estas clases son clases que sólo van a tener una instancia. Estas surgen debido a que varias clases del sistema pueden querer llamar a los métodos de la única instancia de una clase controladora.

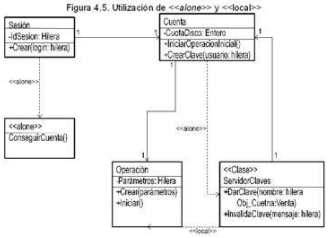
Las clases controladoras se representan utilizando una etiqueta de la clase con el estereotipo <<*alone*>> al igual que las relaciones de las relaciones de dependencia de las clases que lo usan. La figura 4.4. presenta este tipo de clases.

Figura 4.4. Clase controladora

La figura 4.5. muestra un diagrama de clases donde se observa el uso de la etiqueta *alone* y de la etiqueta local.

Figura 71. Páginas 81 – 84 Libro “Programación intermedios”

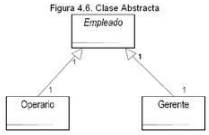
Figura 4.5. Utilización de <<*alone*>> y <<local>>



4.4.4. Clases Abstractas

Las clases abstractas indican que la clase definida no se puede instanciar debido a que posee métodos abstractos. Estas clases se pueden utilizar si se definen subclases que implementen los métodos abstractos definidos. Las clases abstractas se denotan con el nombre de la clase y de los métodos con letra cursiva. La figura 4.6. muestra la representación de una clase abstracta.

Figura 4.6. Clase Abstracta



4.4.5. Cómo Elaborar un Diagrama de Clase

La elaboración de un diagrama de clases debe seguir los siguientes pasos:

- Identificar todas las clases que participan en la solución de software.
- Representar las clases en diagramas de clases.
- Duplicar los atributos provenientes de conceptos creados en el modelo conceptual.
- Añadir los métodos analizando los diagramas de interacción.
- Añadir la información de tipos, la visibilidad y valores iniciales en los atributos.
- Añadir las visibilidades, los parámetros con su tipo y el tipo de retorno (cuando aplique) a métodos.
- Añadir las relaciones de dependencia para indicar visibilidad no relacionada con los atributos.

4.4.6. Métodos

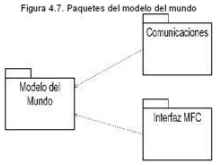
Los métodos que no se incluyen en la descripción del diagrama de clases pueden dividirse en dos formas. El método crear que permite representar la instanciación o inicialización. Los métodos de acceso son aquellos que obtienen o establecen el valor de los atributos. La estructura de estos métodos permite poder acceder y establecer el valor para cada atributo y la declaración private de todos los atributos. Estos métodos de acceso al igual que los de creación no se incluyen en la descripción del diagrama de clases.

4.4.7. Paquetes

Un paquete es el mecanismo por el cual se pueden organizar los elementos en grupos. Los paquetes representan un grupo de elementos del modelo.

Un sistema es considerado como un único paquete, el cual contiene el resto del sistema. Un paquete puede anidarse. Esto permite que un paquete contenga a otro paquete. Los paquetes pueden tener asociaciones de dependencia o de generalización entre ellos. Los paquetes se representan mediante un rectángulo con una pestaña. La figura 4.7. muestra dos paquetes que dependen del paquete modelo del mundo. Los tres modelos poseen su contenido encapsulado.

Figura 4.7. Paquetes del modelo del mundo



Ejemplo 4.2.

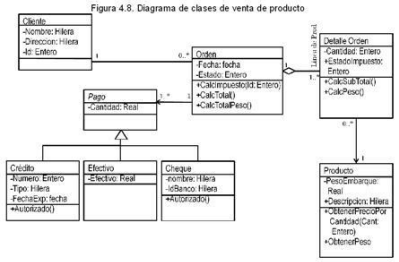
La venta de un producto es el proceso en que un cliente compra cierto producto y proporciona cierto pago por él. El cliente compra el producto y esto queda registrado en una orden de compra. Esta orden de compra también contiene el tipo de pago que realiza el cliente además del detalle de la orden. El cliente puede realizar tres tipos de pago. Estos pagos son en efectivo, con cheque o al crédito. A partir de esto realizar el diagrama de clases correspondiente.

El diagrama de clases se inicia identificando los conceptos que constituyen las clases. Esto se realiza por medio del modelo conceptual. Este caso no posee un modelo conceptual por lo cual se encontrarán las clases a partir del enunciado. Las clases identificadas son: cliente, pago en efectivo, pago con cheque, pago al crédito, producto, orden de compra. Al analizar se poseen tres clases de pago que son específicas. Esto se puede generalizar en una clase pago que posee los tres tipos de pago específicos. Esta clase pago además es abstracta. Esto debido a que la clase pago no se debe instanciar ya que cada clase específica la implementará. La clase orden de pago con producto se relacionan de tal forma que el detalle de la orden es una nueva clase.

Las clases han sido identificadas. El siguiente paso es identificar los atributos para cada una de las clases identificadas y se establece el tipo de visibilidad que contendrán estos atributos. Los métodos de cada clase también deben ser identificados y deben establecerse su visibilidad.

El siguiente paso es relacionar las clases. La clase cliente se relaciona de forma asociativa con la clase orden. La clase crédito, efectivo y cheque se relacionan de forma de generalización con la clase pago. La clase orden se asocia a la clase pago. La clase detalle de orden se relaciona por medio de la composición a la clase orden. La clase detalle de orden se relaciona de forma asociativa a la clase producto. La relación de la clase detalle de orden con la clase orden se le denomina línea de producto (rol). El diagrama finaliza añadiendo la multiplicidad a cada una de las relaciones encontradas. La figura 4.8. presenta el diagrama de clases correspondiente a la venta de un producto.

Figura 4.8. Diagrama de clases de venta de producto



Ejemplo 4.3.

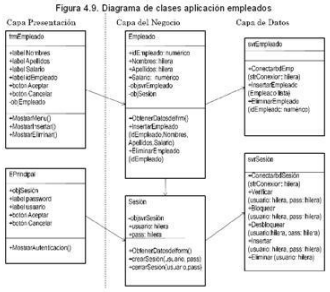
La municipalidad de San Miguel Petapa desea realizar una aplicación que permita añadir y eliminar a sus empleados. Esta aplicación se desarrollará mediante el uso de una arquitectura de tres capas.

La aplicación constará de una interface simple. Esta tendrá dos opciones las cuales serán añadir y eliminar usuario. Las dos opciones dependiendo de la que se elija desplegarán una ventana que permitirá realizar la opción elegida. La aplicación únicamente se podrá utilizar si el usuario es autenticado. Esto quiere decir que en la capa de negocios existe un objeto Sesión. Este objeto permitirá

Figura 72. Páginas 85 – 88 Libro “Programación intermedios”

la autenticación. El objeto de no haber sido creado antes de utilizar alguna opción, desplegará una ventana para autenticar al usuario. Los datos de los empleados serán guardados en una base de datos. Los usuarios serán almacenados en otra base de datos. La autenticación será válida solamente a los usuarios de la base de datos de usuarios. La base de datos bloqueara y desbloqueará al usuario que esté o no utilizando la aplicación. La figura 4.9. muestra este diagrama de clases.

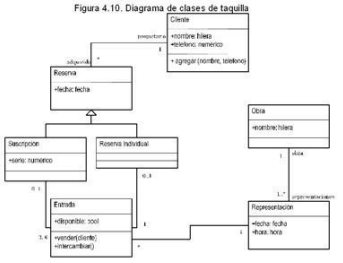
Figura 4.9. Diagrama de clases aplicación empleados



Ejemplo 4.4.

El ejemplo 3.6. muestra la forma de funcionar de una taquilla. Este ejemplo contendrá la parte del dominio de ventas de entradas. Las clases que principales identificadas son cliente, reserva, entrada y representación. Los clientes deben tener muchas reservas. Las reservas son realizadas por un cliente. Las reservas pueden ser una suscripción ó una reserva individual. Estas reservan entradas. Las entradas forman parte de una suscripción o de una reserva individual, pero no pueden pertenecer a ambas. Las representaciones tienen muchas entradas disponibles. Las representaciones se identifican por una obra, una fecha y una hora.

Figura 4.10. Diagrama de clases de taquilla



4.5. Introducción a Diagrama de Colaboración

Los diagramas de colaboración pertenecen a los diagramas de interacción. Los diagramas de colaboración permiten mostrar la interacción que se genera entre los objetos en forma de un grafo o de una red. El diagrama permite observar la interacción de un objeto frente a los demás objetos. Estos diagramas permiten el envío de mensajes por medio de los enlaces que se originan en los objetos.

Los diagramas de colaboración poseen una dependencia del modelo conceptual, los casos de uso reales y el concepto de operación del sistema. Los diagramas de colaboración se caracterizan debido a que brindan excepcional expresividad, la capacidad de comunicar la información y brindan economía de espacio.

Los diagramas de colaboración hacen uso de los conceptos de objetos, enlaces y flujo de mensajes.

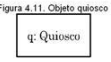
4.5.1. Objeto

Los objetos en el diagrama de colaboración se representan como un rectángulo el cual contiene el nombre y la clase del objeto en el formato: nombreObjeto:nombreClase.

Los objetos compuestos son una representación de un objeto y sus atributos. Los objetos contenidos se muestran dentro del rectángulo que representa al objeto que los contiene.

Un objeto puede ser cualquier instancia de una clase. Un objeto quiosco se muestra en la figura 4.11. El nombre de la clase es Quiosco y el nombre del objeto es q.

Figura 4.11. Objeto quiosco

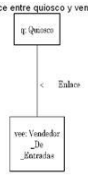


4.5.2. Enlaces

Un enlace es una instancia de una asociación en un diagrama de clases. Los enlaces se representan como una línea continua que une dos objetos, además de un número que indica el orden dentro de la interacción. Los estereotipos pueden ser utilizados para indicar si el objeto que recibe el mensaje es un atributo, parámetro de un mensaje anterior, un objeto local o global.

Un enlace puede darse cuando un objeto quiosco se relaciona con un vendedor de entradas. La figura 4.12. muestra el enlace entre estos objetos.

Figura 4.12. Enlace entre quiosco y vendedor de entradas



4.5.3. Flujo de Mensajes

El flujo de mensajes es utilizado para expresar el envío de un mensaje. Este es representado por una flecha dirigida cerca de un enlace.

La sintaxis para representar a los mensajes puede ser de dos tipos. El primer tipo se utiliza cuando el mensaje no retorna ningún valor. Esta sintaxis es: Nombre_Mensaje(Parámentro1: Tipo, Parámetro2: Tipo...). El segundo tipo se utiliza cuando el mensaje retorna algún valor para la variable de retorno. Esta sintaxis es: VarRetorno = Nombre_Mensaje(Parámetro1: Tipo) Tipo_Retorno.

Los mensajes que surgen en un enlace pueden ser uno o varios. Los mensajes entre el objeto quiosco y vendedor de entradas son: solicitar (cuenta, respresentación), brindar(lista de asientos), comprar(asientos), confirmar (asientos, costo). La figura 4.13. muestra estos mensajes y el flujo de los mismos.

Figura 73. Páginas 89 – 92 Libro “Programación intermedios”

Figura 4.13. Flujo de mensajes entre quiosco y vendedor de entradas

4.5.4. Iteración o Ciclo

La iteración permite la construcción de algo a medida que se avanza. Esto significa que el primer paso podría ser un borrador y este borrador se mejora paso a paso hasta que se obtiene el producto final. Los diagramas de colaboración utilizan la siguiente sintaxis para las iteraciones:

- No_Paso * : [1 = 1... No] mensaje
- No_Paso * : [x > 0] mensaje

Los ciclos pueden surgir dentro de enlaces. Esto quiere decir que si tenemos un objeto p: Producto con un enlace de otro objeto i: Ingrediente y este último es un multi objeto, entonces se puede tener un ciclo. El mensaje para este enlace es servir (valor). Este indica el tipo de ingrediente que se desea añadir en el ciclo. La figura 4.14. muestra esta situación.

Figura 4.14. Flujo de mensajes entre quiosco y vendedor de entradas

4.5.5. Marcadores de Creación y Destrucción de Objetos

Los marcadores se agregan al diagrama conforme los objetos son creados y destruidos. La palabra new o delete son utilizadas para describir estas acciones.

4.5.6. Preparación de Diagramas de Colaboración

Los diagramas de colaboración deben realizarse considerando los siguientes pasos:

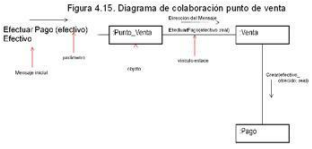
- Elaborar un diagrama para cada operación del sistema durante el ciclo actual de desarrollo, en cada mensaje del sistema.
- Dibujar un diagrama incluyéndolo como mensaje inicial.
- Si el diagrama se torna complejo se debe dividir en diagramas más pequeños.
- Diseñar un sistema de objetos interactivo que realice las tareas usando como punto de partida las descripciones de los casos de uso.
- Se puede aplicar Grasp y otros patrones para desarrollar un buen diseño. Un patrón son directrices y principios estructurales. Grasp es un conjunto de patrones para diagramas de colaboración.

Ejemplo 4.5.

El proceso de venta de un producto conlleva realizar el pago de forma que este puede realizarse en efectivo, por cheque o al crédito. Realizar un diagrama de colaboración considerando la tabla 3.2., que corresponde al pago en efectivo.

La realización del diagrama de colaboración inicia identificando los objetos. Los objetos identificados son tres. Estos son :Punto_Venta, :Venta, :Pago. El proceso comienza con el mensaje inicial Efectuar_Pago_Efectivo. El mensaje lleva un parámetro que sería la cantidad en efectivo a cancelar. Este mensaje llega al objeto :Punto_Venta. :Punto_Venta y :Venta se relacionan por un mensaje que sería Efectuar_Pago. El objeto :Venta y el objeto :Pago se relacionan por otro mensaje que sería Crear. Este mensaje permite crear el pago y mandarle como parámetros la cantidad en efectivo a cancelar. La figura 4.15. muestra el diagrama de colaboración correspondiente.

Figura 4.15. Diagrama de colaboración punto de venta



Ejemplo 4.6.

El ejemplo 4.6. se fundamenta en el diagrama de clases de la figura 4.10. del ejemplo 4.4. Este ejemplo muestra el diagrama de clases del sistema de taquilla en lo que respecta a la venta de las entradas. El diagrama de colaboración muestra entonces la interacción para reservar entradas. La petición es obtenida mediante el quiosco. Esta es utilizada para encontrar la base de datos de la representación deseada. El vendedor de entradas pide un número de asientos para la representación. Los asientos se encuentran en un variado de precio y es bloqueado temporalmente. Esto se devuelve al quiosco

de selección de clientes. El cliente es el que realiza la selección de la lista de asientos, este obtiene los asientos seleccionados y el resto son liberados.

Figura 4.16. Diagrama de colaboración de Taquilla

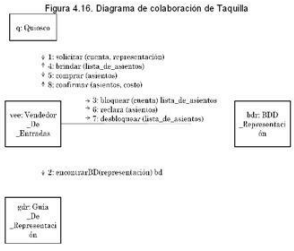


Figura 74. Páginas 93 – 96 Libro “Programación intermedios”

5. INTEGRACIÓN DE ARTEFACTOS DE SOFTWARE

5.1. Introducción a Diagramas de Actividades

Los diagramas de actividades representan el aspecto dinámico del sistema. Estos son un caso especial de los diagramas de estados que muestran estados de acción. Estos diagramas son diagramas de flujo. Esto es debido a que gráficamente son un conjunto de nodos y arcos. En los diagramas de actividades las transiciones son enviadas automáticamente al termina la acción ejecutada.

Los diagramas de actividades pueden dar detalle de un caso de uso, objeto o un mensaje de un objeto. Estos diagramas sirven para representar las transiciones internas. Los diagramas de actividades generalmente modelan los pasos de un algoritmo. Esto es debido a que muestran la concurrencia, los ciclos y las condiciones.

Los diagramas de actividades conceptualmente muestran como fluye el control de unas clases a otras clases con funcionalidad de cambiar con un flujo total que corresponde con la consecución de un proceso más complejo.

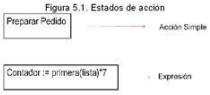
Los elementos básicos de un diagrama de actividades son:

- Estados de actividad.
- Estados de acción.
- Transiciones.
- Objetos.

5.1.1. Estado de Acción

Los estados de acción son atómicos. Esto significa que la ejecución puede ser considerada instantánea y no se puede interrumpir. Los estados se representan con un rectángulo. La figura 5.1. presenta una acción simple de un estado de acción y una expresión de un estado. Estas dos son consideradas como estados de acción.

Figura 5.1. Estados de acción



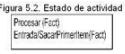
5.1.2. Estado de Actividad

Una actividad es una sucesión de sub actividades y acciones. Las actividades se representan con un rectángulo. Este rectángulo puede estar o no con bordes redondeados.

Los estados de actividad se pueden descomponer en sub actividades representadas a través de otro diagrama de actividades. Estos estados pueden ser interrumpidos y tardan cierto tiempo en completarse.

Los estados de actividad pueden contener tanto acciones de entrada como acciones de salida. La figura 5.2. muestra un estado de actividad. Este estado representa el Procesar una factura. Este mismo presenta las acciones de entrada y salida que serían sacar el primer objeto.

Figura 5.2. Estado de actividad



Los estados de actividad y de acción pueden poseer parámetros ó retorno de valores. Estos estados pueden poseer pre condiciones y post condiciones. Estas se representan por medio de corchetes “[“, “]”.

Las actividades pueden ser agrupadas. Estas agrupaciones se dan conforme al contexto en que las actividades son realizadas. Las actividades pueden pertenecer o no a un solo contexto. Esto permite que se creen *swinlanes*. Estas permiten poseer varios contextos en un diagrama de

actividades. Los *swinlanes* pueden ser divididos por un nombre en la parte superior y una línea punteada o sólida para dividir cada *swinlanes*.

5.1.3. Transiciones

Las transiciones son utilizadas para reflejar el paso de un estado hacia otro estado. Las transiciones son representadas por medio de una flecha. La figura 5.3. muestra como se observa una transición.

Figura 5.3. Transición

5.1.4. Bifurcaciones

Las bifurcaciones representan caminos alternativos. Estos tienen una transición de entrada y dos o más transiciones de salida. Las transiciones de salida requieren expresiones booleanas que se evaluarán al llegar a la bifurcación. Esto para cada transición de salida. Las condiciones de la bifurcación deben ser excluyentes y completar todos los casos. Las bifurcaciones se representan por medio de un rombo lleno. Las bifurcaciones pueden ser salientes y entrantes.

La figura 5.4. representa la forma en que se utilizaría una bifurcación. Esta imagen presenta una solicitud de materiales. Esta presenta una bifurcación donde si hay materiales disponibles se realiza la entrega de materiales, si no hay materiales disponibles se hace un pedido y si no se realiza un estado no identificado.

Figura 5.4. Bifurcación

Ejemplo 5.1.

El proceso de compras dentro de una empresa es una serie de pasos. Los pasos que permiten realizar una compra dentro de una empresa son:

- Realizar una cotización.
- Elegir la mejor oferta.
- Crear una orden de compra.
- Autorizar la orden de compra.
- Verificar la contabilidad y presupuesto.
- Solicitar al proveedor de productos.

Los pasos anteriores ocurren dentro del departamento de compras y dentro del departamento de finanzas (presupuesto). A partir de estos pasos realizar el diagrama de actividades correspondiente.

Los pasos citados corresponden a estados de actividades. La identificación de actividades resulta sencilla debido a que ya han sido identificadas en el enunciado. La actividad de verificar presupuesto y contabilidad son dos actividades que se realizan una después de la otra.

Las actividades que corresponden al departamento de presupuesto son la de verificar contabilidad y la de verificar presupuesto. Las otras actividades pertenecen al departamento de compras.

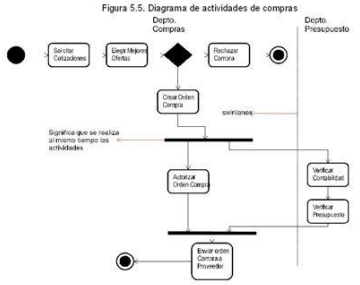
Figura 75. Páginas 97 – 100 Libro “Programación intermedios”

El diagrama debe comenzar con la actividad de solicitar cotización. Esta actividad se realiza hasta elegir las mejores ofertas. Este punto permite una bifurcación donde si no existe ninguna buena oferta se rechace la compra. Este punto también permite que si existe una buena se cree la orden de compra.

La actividad de crear la orden de compra permite que dos actividades se realicen al mismo tiempo. La notación de esto es mediante una linea horizontal gruesa. Esta permite indicar que las actividades se realizarán al mismo tiempo. Las actividades a realizar serán autorizar la orden de compra y verificar la contabilidad. Esta última actividad permite verificar el presupuesto. Las actividades verificar presupuesto y autorizar la orden de compra convergen en la actividad enviar orden de compra al proveedor. La linea gruesa sólida permite ser utilizada para representar esta convergencia, debido a que ambas actividades llegan al mismo tiempo.

La figura 5.5. muestra como quedaria representado gráficamente el diagrama de actividades para realizar una compra en una empresa.

Figura 5.5. Diagrama de actividades de compras



5.2. Modelado Físico de un Sistema

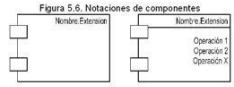
5.2.1. Componente

Los componentes son partes independientes de un sistema. Esto es conforme el punto de vista de un usuario final. Los componentes pertenecen al mundo físico. Los componentes pueden ser también un conjunto de clases de un diagrama de clases. Estos representan un bloque de construcción.

El componente debe definir la abstracción precisa con una interfaz bien definida y debe permitir reemplazar fácilmente los componentes más viejos con otros más nuevos y compatibles.

La notación de los componentes corresponde a un rectángulo con dos rectángulos pequeños uno encima de otro en la parte izquierda. Los componentes pueden ser denotados de dos formas. La primera una forma simple que posee solamente el nombre del componente y su extensión "nombre","extensión". La segunda una forma extendida donde además del nombre contiene las operaciones que realiza el componente. La figura 5.6. muestra ambas notaciones para un componente.

Figura 5.6. Notaciones de componentes



Los componentes pueden ser clasificados en tres tipos:

- Componentes de despliegue: estos son los componentes necesarios y suficientes para formar un sistema ejecutable. La extensión de estos componentes son .dll, .exe, etc.
- Componentes producto del trabajo: estos componentes son productos que quedan al final del desarrollo de una aplicación. Estos componentes pueden ser código fuente, archivos de configuración, etc.
- Componentes de ejecución: estos componentes se crean como consecuencia de un sistema de ejecución. Estos son objetos COM+ instanciados a partir de un dll.

Los componentes pueden encontrarse compuestos por otros componentes. Esto significa que los componentes pueden agruparse. La agrupación de

componentes se denomina paquete. Los componentes agrupados pueden especificar relaciones de dependencia, generalización, asociación y realización.

Los componentes poseen los siguientes estereotipos:

- <<*executable*>>: componente que se puede ejecutar en un nodo.
- <<*library*>>: biblioteca de objetos.
- <<*document*>>: representa un documento.
- <<*table*>>: una tabla en la base de datos.
- <<*file*>>: representa código fuente.

5.2.2. Interfaces

Las interfaces son servicios propios de un componente. Estos son lazos de unión entre unos componentes y otros. Esto significa que las interfaces permiten conectar un componente con otro componente.

Las interfaces se representan como un circulo entre los componentes. Estos se unen por una linea sólida y por una linea dirigida punteada. La parte superior del circulo contiene el nombre de la interfaz. La figura 5.7. presenta dos componentes conectadas por medio de una interfaz.

Figura 5.7. Notación de interfaz

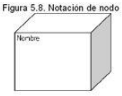
5.2.3. Nodo

Un nodo es un elemento físico el cual existen en tiempo de ejecución. Los nodos representan un recurso computacional que usualmente tiene alguna memoria y capacidad de procesamiento.

Los nodos sirven para modelar la topología del hardware sobre el que se ejecuta el sistema. Un nodo debe tener un nombre que lo distinga del resto de nodos. Los nodos pueden participar en relaciones de dependencia, generalización y asociación.

Los nodos son representados gráficamente por medio de un cubo. Este cubo posee en la esquina superior izquierda el nombre del nodo. La figura 5.8. muestra la notación de un nodo.

Figura 5.8. Notación de nodo



5.3. Introducción a Diagramas de Implementación

Los diagramas de implementación permiten considerar la estructura del código y la estructura del sistema donde este código es ejecutado. Estos diagramas comprenden los diagramas de componentes y los diagramas de ejecución o despliegue.

5.3.1. Diagrama de Componentes

El diagrama de componentes se utiliza para modelar la vista estática de un sistema. Este diagrama muestra las dependencias entre componentes. El diagrama de componentes muestra también las librerias, tablas, archivos, ejecutables, y documentos que forman parte del sistema.

Los diagramas de componentes pueden modelar la estructura del código fuente y la interacción entre ejecutables. Los estándares para representar los diagramas de colaboración corresponden en *UML* al *Executable Library Table File Document*. Este estándar se ha denotado en los primeros incisos de la unidad 5.2. El estándar de *UML* no es suficiente en caso se desee modelar sistemas en internet. Estos utilizan componentes *ASP*, *HTML* y *Scripts*. Esto permite que se utilice el estereotipo *Web Applications Extension* (*WAE*).

Ejemplo 5.2.

Un programa de contabilidad dentro de área local, posee varios componentes para que pueda funcionar correctamente. El programa posee un componente que permite realizar la comunicación por medio del protocolo *TCP*

Figura 76. Páginas 101 – 104 Libro “Programación intermedios”

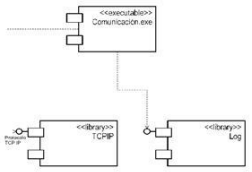
IP. Este componente se encuentra integrado por un ejecutable. Este ejecutable se llama comunicación.exe.

El ejecutable comunicación.exe permite realizar la comunicación dentro del sistema. Un log permite guardar todo lo que ocurra mientras se realiza la comunicación. Realizar el diagrama de componentes para el componente de comunicación.

El enunciado muestra que existen tres componentes identificados fácilmente. Estos son el archivo ejecutable que se llama comunicación.exe que es un componente ejecutable, el archivo que sirve de log y la librería que permite la comunicación *TCP IP* estos son componentes del tipo librería. El componente ejecutable utiliza los otros dos componentes por medio de su interfaz.

La figura 5.9. muestra el diagrama de componentes correspondiente al programa de contabilidad en su parte de ejecución de la comunicación para el sistema.

Figura 5.9. Diagrama de componentes aplicación de comunicación

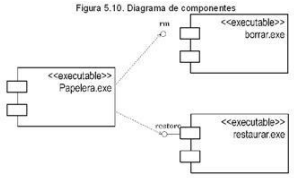


Ejemplo 5.3.

Una aplicación para Linux que utilice el Shell para eliminar los archivos. Esto existe actualmente, pero no posee función de papelera de reciclaje. La papelera de reciclaje será entonces una aplicación que permita eliminar un archivo mediante la Shell y que no sea eliminado de forma permanente. Esto quiere decir que el archivo eliminado se podrá posteriormente restaurar o eliminar definitivamente según sea el caso.

Una solución a este problema es realizar una aplicación que sea capaz de interceptar el módulo rm (borrar archivo) y sustituirlo por un módulo propio que tenga la función de papelera. Esta aplicación también debe sustituir el módulo restore (restaurar) para poder restaurar el archivo o archivos eliminados. La aplicación consiste en un programa principal que realiza la sustitución de los módulos. Esta aplicación utiliza entonces las aplicaciones borrar y la de restaurar. La figura 5.10. muestra el diagrama de componentes correspondiente a la aplicación de la papelera de reciclaje.

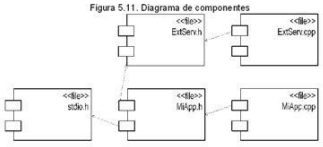
Figura 5.10. Diagrama de componentes



Ejemplo 5.4.

Una aplicación realizada en C utiliza dos archivos .cpp. Esta también utiliza tres librerías .h. El archivo cpp principal hace uso de las librerías para poder funcionar. La disposición se muestra en la figura 5.11. El archivo MiApp.cpp utiliza su librería MiApp.h. Esta librería utiliza a su vez la librería stdio.h y ExtServ.h. Esta última librería es utilizada por el archivo ExtServ.cpp.

Figura 5.11. Diagrama de componentes



5.3.2. Diagrama de Despliegue ó Ejecución

Los diagramas de despliegue y ejecución básicamente son lo mismo. Estos buscan representar los componentes que deben estar instalados y los nodos que forman la infraestructura donde corren dichos componentes. Estos diagramas utilizan la misma notación. La diferencia radica solamente en el nombre debido a que se utiliza despliegue cuando se refiere a aplicaciones WEB debido a que estas se despliegan en un browser. La palabra ejecución es utilizada generalmente cuando son aplicaciones cliente – servidor. Esto es debido a que las aplicaciones son ejecutadas.

Los diagramas de ejecución indican la situación física de los componentes lógicos desarrollados. Estos diagramas sitúan el software en el hardware que lo contiene. Los diagramas de ejecución se consideran como una capa componente de hardware. Estos diagramas se representan como un nodo.

Los diagramas de ejecución muestran la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software, procesos y objetos que se ejecutan en ellos. Las instancias de componentes de software pueden estar unidas por relaciones de dependencia, posiblemente interfaces.

Un diagrama de ejecución se considera como un grafo de nodos unidos por medio de conexiones de comunicación. El diagrama de ejecución generalmente implica modelar la topología del hardware sobre el cual se ejecuta el sistema.

Ejemplo 5.5.

Un sistema corporativo que permite que sus empleados puedan realizar búsquedas por medio de una comunicación vía LAN, es utilizado en una arquitectura de dos capas. Esto significa que utiliza la arquitectura cliente servidor. El servidor se encuentra compuesto por varios ejecutables que permiten iniciar el directorio de la corporación y realizar las búsquedas. El cliente utiliza un programa de presentación donde pueden solicitar al servidor que devuelva sus consultas. Realizar el diagrama de ejecución correspondiente.

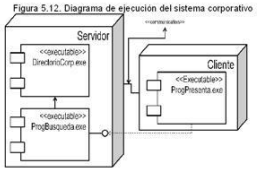
La identificación de los nodos es sencilla. Esto debido a que el enunciado brinda la información necesaria para identificarlos. Los nodos son el servidor y el cliente.

El servidor cuenta con los dos ejecutables que permiten realizar y mantener el sistema. Estos ejecutables son los archivos DirectorioCorp.exe y ProgBusqueda.exe. Esto permite indicar que existen dos componentes que se relacionan entre si dentro del servidor.

El cliente cuenta con un ejecutable que le permite ver el resultado de su búsqueda. Este ejecutable se denomina ProgPresenta.exe.

El ejecutable del cliente se relaciona mediante la interfaz que posee el componente ProgBusqueda.exe. La figura 5.12. muestra el diagrama de ejecución correspondiente al sistema de búsqueda corporativo.

Figura 5.12. Diagrama de ejecución del sistema corporativo



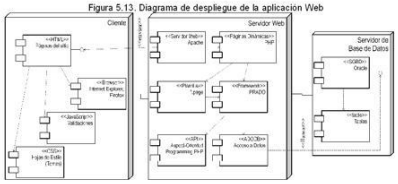
Ejemplo 5.6.

Una empresa desea realizar una aplicación para una institución. El sistema se ha decidido realizarlo en una arquitectura de tres capas. El proceso del sistema indica que el usuario puede acceder al sistema por medio de una URL desde un *browser*. El *browser* se encarga de cargar las páginas del sitio web. Estas a su vez utilizan los css y los javascript.

La aplicación de despliegue utiliza una arquitectura un servidor web tipo apache. El framework que se utiliza es un PRADO (Php *Rapid Application Development Object-Oriented*) que se basa en el principio de inversión de control. Este framework carga los archivos php que poseen la lógica de negocio y el acceso a datos utilizando una plantilla *.page. El sevidor de datos se realizará sobre el DBMS Oracle. La figura 5.13. muestra el diagrama de despliegue de esta aplicación web.

Figura 77. Página 105 Libro “Programación intermedios”

Figura 5.13. Diagrama de despliegue de la aplicación Web



C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2
CLASE LIBRO

6. APENDICE

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA Página | 106
FACULTAD DE INGENIERÍA ESCUELA CIENCIAS Y SISTEMAS

C771 INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 2
CLASE LIBRO

C771 UNIVERSIDAD DE SAN CARLOS DE GUATEMALA Página | 107
FACULTAD DE INGENIERÍA ESCUELA CIENCIAS Y SISTEMAS

8. DOCUMENTACIÓN DE APOYO DEL CURSO DE SISTEMAS DE BASES DE DATOS 1

8.1. Descripción

Documentación de apoyo del curso de Sistemas de Bases de Datos 1, el cual fue realizado en formato de un libro que contiene las unidades y temas más importantes impartidos en el curso.

El libro consta de tres capítulos. Estos capítulos son:

a. Introducción a conceptos de bases de datos.
b. Introducción al modelado entidad-relación.
c. Introducción a SQL.

Figura 78. Páginas 1 – 4 Libro “Bases de datos principiantes”

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

ESCUELA DE CIENCIAS Y SISTEMAS

LIBRO

NOMBRE DE CURSO: Sistemas de Bases de Datos 1

CODIGO DEL CURSO: 0774

NOMBRE DEL MODULO: Bases de Datos Principiantes

Autor: Mario José Bautista Fuentes

CONTENIDO

1. INTRODUCCIÓN A CONCEPTOS DE BASES DE DATOS

1.1. Conceptos Básicos de Bases de Datos

1.1.1. Modelo

La representación de algo que existe en la vida real es básicamente un modelo. Un modelo es una abstracción simplificada de la realidad.

1.1.2. Base de Datos

Una base de datos es un conjunto de datos relacionados entre sí, que reúne dos condiciones:

- La primera condición es ser persistentes. Esto quiere decir que los datos deben ser almacenados para poder ser utilizados posteriormente.
- La segunda condición es ser utilizados por una aplicación de software. Esto indica que los datos deben estar de tal forma que una aplicación puede interpretarlos y realizar operaciones con los datos.

1.1.3. Sistema Administrador de Base de Datos

El sistema administrador de base de datos es una herramienta de tecnología que soporta a la base de datos. Las siglas en ingles son DBMS (*data base management system*). Actualmente los DBMS más importantes son:

- Oracle
- SQL Server
- My SQL
- PostgreSQL

El DBMS se encuentra integrado por una serie de funciones. Las funciones principales que debe de poseer se encuentran:

- Definición de datos: esto significa que el DBMS es capaz de la aceptación de definición de datos tales como esquemas externos, el esquema conceptual, el esquema interno y las transformaciones pertinentes. El DBMS contiene entre sus componentes un procesador ó compilador DDL para cada lenguaje de definición de datos.
- Manipulación de datos: el DBMS debe contener un procesador o componente DML para tratar con el lenguaje de manipulación de datos. Esto le sirve al DBMS para poder manipular las peticiones de recuperación, actualización o eliminación de datos existentes en la base de datos o para la agregación de nuevos datos en la base de datos. Las peticiones DML se dividen en dos. La primera petición es una petición planeada Esta se caracteriza debido a que su necesidad se prevé antes de que se ejecute la petición. La segunda petición es una petición no planeada. Esta es una consulta ad hoc. La necesidad no se ve hasta el momento en que se realiza la petición.
- Optimización y ejecución: el optimizador se encarga de las peticiones planeadas y no planeadas. El optimizador determina la forma más eficiente de implementar la petición.
- Seguridad e integridad de los datos: el DBMS es el encargado del control de las peticiones de usuario y es el encargado del rechazo de cualquier intento de violación a las restricciones de seguridad y de integridad impuestas por el DBA.
- Recuperación de datos y concurrencia: el administrador de transacciones o monitor de transacciones es el encargado de administrar las transacciones. Este impone controles de recuperación y de concurrencia.
- Diccionario de datos: esta función debe ser proporcionada por el DBMS. El diccionario de datos es visto como una BD que contiene datos acerca de los datos. Estos datos acerca de los datos se conocen como metadatos o descriptores. El diccionario de datos contiene entonces definiciones de otros objetos del sistema.
- Rendimiento: el DBMS debe ser capaz de realizar todas las funciones descritas anteriormente de la forma más eficiente posible.

1.1.3.1. Usuario

Un usuario es la persona que utiliza las aplicaciones y la base de datos. El usuario puede ser:

- Programador de la aplicación: este usuario crea la aplicación que permite hacer uso de la base de datos. El programador de la aplicación no es el encargado de la base de datos.

Figura 79. Páginas 5 – 8 Libro “Bases de datos principiantes”

- Usuario final: este usuario hace uso de la aplicación. El usuario final no comprende cómo la aplicación interactúa con la base de datos. Este usuario solamente utiliza la aplicación.
- DBA (administrador de base de datos): este usuario administra la plataforma electrónica de la base de datos. El DBA es el responsable de los datos que se almacenan en la base de datos y de la estructura que estos poseen.

1.1.4. Arquitectura de los Sistemas de Bases de Datos

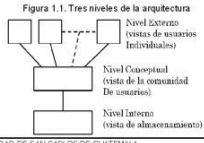
La arquitectura fue propuesta por el Grupo de Estudio en Sistemas de Administración de Base de Datos (ANSI/SPARC). Esta arquitectura es conocida por lo tanto como arquitectura ANSI/SPARC.

La arquitectura es dividida en tres niveles. Estos niveles son:

- Nivel interno: este nivel se conoce como nivel físico. El nivel interno es el que se encuentra más cercano al almacenamiento físico. Esto significa que el nivel corresponde a como los datos se encuentran almacenados físicamente.
- Nivel externo: este nivel se conoce como nivel lógico de usuario. El nivel externo corresponde a la forma en que los datos son percibidos por los usuarios.
- Nivel conceptual: este nivel se conoce de dos formas. La primera como el nivel lógico de la comunidad y la segunda como nivel lógico. El nivel conceptual es un nivel de in-dirección entre el nivel interno y el nivel externo.

Los tres niveles se muestran en la figura 1.1. Esta figura presenta la forma de interacción que existe entre los tres niveles.

Figura 1.1. Tres niveles de la arquitectura



2. INTRODUCCIÓN A MODELADO ENTIDAD RELACION

2.1. Base de Datos Relacional

Una base de datos relacional es una base de datos (DB), la cual se fundamenta en el modelo relacional. Estas bases de datos son percibidas como si fueran un conjunto de tablas.

2.2. Modelo Relacional

El modelo relacional fue creado por Edgar F. Codd. Este modelo permite que los datos se estructuren a nivel lógico como tablas formadas por filas y columnas. Esta estructura a nivel físico es completamente diferente.

El modelo relacional maneja 3 aspectos:

- La estructura de datos: esto significa que los datos en el modelo relacional se encuentran estructurados y debido a esto garantiza su acceso.
- La integridad de datos: esto significa que los datos en la base de datos deben ser correctos. Los datos dentro de la base de datos harán referencias a datos existentes y en ningún momento referenciarán datos falsos o inexistentes. Asimismo ninguna tupla en la base de datos serán repetidos.
- El manejo de datos: esto significa el conjunto de operaciones que permiten manipular los datos.

El modelo entidad relación es forma en que se expresa gráficamente un modelo de datos. El diagramar un modelo entidad relación se encuentra sujeto a una serie de convenciones que se deben seguir para realizarlo.

Un modelo de datos se encuentra compuesto de entidades (tablas en la base de datos) y las relaciones entre estas entidades.

2.2.1. Entidad

Una entidad es algo que se puede identificar de manera única y sobre la cual se requiere llevar algún control.

Una entidad representa cualquier objeto distinguible en un modelo de negocios que se debe representar en la base de datos.

Una entidad se encuentra representada en la Figura 2.1. En esta figura se presenta la notación.

Figura 2.1. Entidad

Las reglas para representar a las entidades son:

- Utilizar una caja con bordes redondeados (softbox).
- La entidad debe tener un nombre único en singular y en minúsculas.
- El sinónimo (si lo hay) debe colocarse entre paréntesis abajo del nombre de la entidad.

Las entidades pueden ser de dos tipos. El primer tipo es una entidad fuerte. Esta entidad existe por si misma. Esto quiere decir que no depende de la existencia de ninguna otra entidad. Estas entidades poseen atributos propios. La entidad persona es un ejemplo de este tipo de entidades. El segundo tipo es una entidad débil. Esta entidad depende de la existencia de otra entidad. Las entidades débiles pueden poseer atributos que identifiquen una llave foránea como única dentro de este tipo de entidad. La entidad ocupación es un ejemplo de entidades débiles.

2.2.1.1. Propiedades ó Atributos

Las entidades se encuentran compuestas por propiedades o atributos. Los atributos son la unidad fundamental que describe un dato. Una entidad se

encuentra integrada por varios atributos. Existen varios tipos de datos para los atributos, entre estos:

- Numérico.
- Hilera (*String*).
- Fecha.

Las propiedades ó atributos poseen características que las identifican. Estas características son:

- Simples o compuestas: las propiedades simples son atómicos. Esto quiere decir que no pueden ser divididas en otras propiedades. Las propiedades compuestas son propiedades integradas por otras propiedades y se pueden dividir pero no es aconsejable hacerlo.
- Identificador único: esta propiedad puede identificar a la entidad y a los demás conjuntos de propiedades.
- Uni-valuados o multi-valuados: las propiedades uni-valuadas son atributos que en el transcurso del tiempo sólo tomarán un valor para una entidad. La cédula es un ejemplo de este tipo de propiedades. Las propiedades multi-valuadas son atributos que en el transcurso del tiempo pueden tener un conjunto de valores para una entidad en particular. El grado académico, el sexo y el estado civil de una persona son ejemplos de este tipo de atributos.
- Opcionales: estas propiedades pueden tomar el valor nulo en cualquier momento.
- Base o derivadas: el valor de estas propiedades depende de otro atributo o entidad. Un ejemplo de este tipo es el salario.

El diagramar las propiedades corresponde a seguir una serie de reglas. Estas reglas son:

- Escribir el nombre de la propiedad o atributo en singular y en letra minúscula.
- El nombre de la entidad no se debe repetir en el nombre del atributo. El nombre del atributo debe identificar claramente el mismo.
- Los atributos obligatorios se marcan con un asterisco (*). Los atributos opcionales se marcan con un círculo (°).
- El identificador único primario se marca con numeral (#). Estos atributos se conocen como llave primaria.
- Un identificador único secundario se marca con un numeral dentro de paréntesis ((#)). Este se conoce como llave alterna.

Figura 80. Páginas 9 – 12 Libro “Bases de datos principiantes”

2.2.1.2. Tupla

Una tupla es considerada como el conjunto de atributos que describen una entidad.

Ejemplo 2.1.

Se desea crear una entidad que corresponda a un curso y establecer sus atributos. Para realizar esta entidad, se identifica que la tupla de la entidad se encuentra compuesta por tres atributos: código, nombre y abreviatura. La Figura 2.2. indica la notación de la entidad curso y de la tupla.

Figura 2.2. Entidad curso

La Figura 2.3. representa como se vería la entidad curso desde el punto de vista físico y no lógico. En la representación física cada fila de la tabla representa una tupla y cada tupla corresponde a una serie de datos específicos y únicos. Estos datos como tupla no pueden ser iguales debido a que se estaría violando la integridad de los datos. Esto debido a que habría datos repetidos. El modelo relacional no concibe que en ningún momento en las entidades se puedan albergar tuplas repetidas.

Figura 2.3. Representación física de la entidad curso

CURSO

codigo	nombre	abreviatura
0771	Progra 2	IPC2
0772	Estructuras	ED

Tupla

2.2.1.3. Llaves Primarias

Una llave primaria es el identificador único para cada tupla de una entidad. La llave primaria puede ser uno o más atributos que hacen única la tupla. Estos atributos son obligatorios. Esta llave identifica a una tupla y la hace única de las

demás tuplas dentro de una entidad. La Figura 2.4. presenta la notación y especifica la llave primaria dentro de una entidad.

Figura 2.4. Llave primaria de curso

2.2.1.4. Llaves Foráneas

Una llave foránea es el conjunto de uno o más atributos que permiten relacionar a la entidad a la que pertenece con la entidad que se relaciona la misma. Las llaves foráneas son referencias de una entidad hacia otras.

Ejemplo 2.2.

Dos entidades, entre ellas estudiante y curso se relacionan de tal forma que un estudiante pude asignarse varios cursos. Asimismo un curso puede ser asignado por un estudiante varias veces con el transcurso de los semestres. La Figura 2.5. representa la relación asignación entre las entidades estudiante y curso. La identificación de las tuplas para cada entidad es lo más importante. Esto permite que se identifique para la entidad asignación dos atributos. Estos atributos forman parte de la tupla de la entidad asignación. Los atributos corresponden a la llave foránea de la entidad estudiante y la llave foránea de la entidad curso. Estas llaves foráneas hacen referencia a las llaves primarias de las entidades citadas. La tupla de la entidad asignación se encuentra compuesta por:

- no asignacion
- estado
- semestre
- carnet (llave foránea de entidad estudiante)
- codigo (llave foránea de entidad curso)

Figura 2.5. Relación asignación

2.2.2. Relaciones Básicas entre Entidades

Una relación es aquella que permite describir una correspondencia entre los datos. Estas surgen debido a que una entidad puede relacionarse con una o varias entidades. Las entidades deben poseer un campo en común para poder establecer un vínculo entre ellas. La llave primaria habitualmente se encuentra contenida en un atributo de la otra entidad.

La cardinalidad es el número de instancias o de elementos de una entidad que pueden ser asociados a un elemento de la otra entidad relacionada. La cardinalidad es representada mediante una pareja de datos de la forma (cardinalidad mínima, cardinalidad máxima). Las posibles parejas son:

- (0,1)
- (1,1)
- (0,M)
- (1,M)
- (M,M)

Las relaciones entre las entidades se encuentran definidas tomando los máximos de cardinalidad que intervienen en la relación. Estas pueden ser:

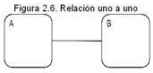
- De uno a uno.
- De uno a muchos.
- De muchos a muchos.

Las relaciones gráficamente se representan por medio de una línea que une a dos entidades.

2.2.2.1. De Uno a Uno

La relación de uno a uno ocurre cuando una instancia en la entidad A se encuentra relacionada únicamente a una instancia de la entidad B. La Figura 2.6. presenta la notación de una relación de este tipo.

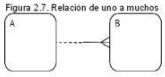
Figura 2.6. Relación uno a uno



2.2.2.2. De Uno a Muchos

Una relación de uno a muchos ocurre cuando una instancia en la entidad A se encuentra relacionada con varias instancias en la entidad B. La Figura 2.7. representa la notación de una entidad de este tipo.

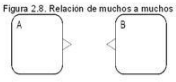
Figura 2.7. Relación de uno a muchos



2.2.2.3. De Muchos a Muchos

Una relación de muchos a muchos ocurre cuando una instancia en la entidad A se encuentra relacionada con varias instancias en la entidad B. La Figura 2.8. representa la notación de una entidad de este tipo.

Figura 2.8. Relación de muchos a muchos



2.2.2.4. Opcionalidad en las Relaciones

Las relaciones pueden ser opcionales u obligatorias. Esto es debido a que en la tupla algunos datos siempre deben aparecer como es el caso de la llave primaria.

Figura 81. Páginas 13 – 16 Libro “Bases de datos principiantes”

Las relaciones opcionales se representan por medio de lineas punteadas. La figura 2.9. representa la notación de una relación opcional.

Figura 2.9. Relación opcional

Las relaciones obligatorias se representan por medio de lineas continuas. La Figura 2.10. representa la notación de una relación obligatoria.

Figura 2.10. Relación obligatoria

La relación se representa por una línea que une dos entidades. Esto quiere decir que la mitad de la línea corresponde a una entidad y la otra mitad corresponde a la otra entidad. El extremo que se encuentra en A indica la visibilidad y la multiplicidad de la entidad A. El extremo que se encuentra en B indica la visibilidad y la multiplicidad de la entidad B.

Las relaciones de uno a muchos (1:M) pueden ser de dos tipos:

- La Figura 2.11. presenta las relaciones más comunes de 1:M.

Figura 2.11. Relaciones Tipo 1, 1:M

- La Figura 2.12. presenta las relaciones menos comunes (si se tiene una de estas se recomienda revisar el modelo e intentar eliminarlas):

Figura 2.12. Relaciones tipo 2, 1:M

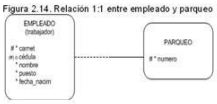
Generalmente las relaciones de muchos a muchos (M:M), surgen inicialmente en un modelo pero deben de ser eliminadas. Estas relaciones posteriormente dan paso a relaciones 1:M. La figura 2.13. presenta la notación de relaciones M:M.

Figura 2.13. Relaciones M:M

Ejemplo 2.3.

La empresa TATA posee una sistema de parqueos para sus empleados. Este sistema permite asociar a cada uno de los empleados un parqueo siempre a un empleado. El empleado siempre poseerá su parqueo y nunca lo perderá a menos que sea despedido de la empresa. El enunciado permite identificar dos tipos de entidades. La entidad empleado y la entidad parqueo. Esto permite establecer la relación de uno a uno entre estas entidades. La opcionalidad parte en que un parqueo no aparece a menos que un empleado esté asociado. Esto permite indicar que un empleado puede poseer un parqueo y que un parqueo debe pertenecer a un empleado. La figura 2.14. representa esta relación.

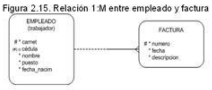
Figura 2.14. Relación 1:1 entre empleado y parqueo



Ejemplo 2.4.

Una empresa de productos posee una facturación en la cual un empleado puede realizar una o más facturas durante el dia. Esto permite establecer que las dos entidades sean empleado y factura. La relación existente entre estas entidades es de uno a muchos. La opcionalidad de esta relación es opcional del lado de uno y totalmente obligatoria de lado de muchos. Esto permite indicar que un empleado puede realizar una o más facturas pero una o más facturas deben ser hechas por solamente un empleado. La figura 2.15. representa la relación de 1:M.

Figura 2.15. Relación 1:M entre empleado y factura



Ejemplo 2.5.

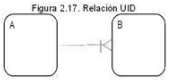
La atención de clientes en una empresa es primordial. Los clientes llegan a la empresa y pueden ser atendidos por uno o más empleados. Los empleados también pueden atender uno o más clientes. La relación permite identificar las entidades empleado y cliente. La relación que surge entre estas entidades es de muchos a muchos. La figura 2.16. representa este tipo de relaciones.

Figura 2.16. Relación M:M entre empleado y entidad

2.2.2.5. Relaciones como UID

UID significa *unique identificator*. Las relaciones como UID se fundamentan en que un campo de la entidad A forma parte de la llave primaria de la entidad B. La Figura 2.17. representa la notación de una relación UID.

Figura 2.17. Relación UID

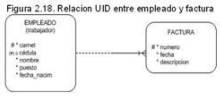


Ejemplo 2.6.

Un empleado puede realizar una o más facturas y varias facturas deben ser realizadas por un empleado. Esta relación se le puede incluir además que la

llave primaria de empleado, sea además de foránea en la entidad factura, parte de la llave primaria de esta entidad. La entidad factura tiene una llave primaria compuesta, formada por el campo número y también por el campo carnet que es la llave primaria de la entidad empleado. La figura 2.18 muestra esta relación.

Figura 2.18. Relacion UID entre empleado y factura



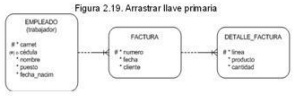
2.2.2.6. Arrastrar la Llave Primaria

La llave primaria de una entidad como se ve en el ejemplo 2.6. puede pasarse como llave foránea hacia otra entidad y formar parte de la llave primaria de esta entidad. Esto permite indicar que si esta entidad que posee una llave compuesta se relaciona por UID a otra entidad, pasará su llave compuesta como foránea y formará una llave compuesta por tres atributos en la entidad que las recibe.

Ejemplo 2.7.

El caso descrito en el ejemplo 2.6. puede extenderse de tal forma que existe una nueva entidad que se denomina detalle_factura. Esta entidad se encuentra relacionada con factura de uno a muchos y de forma UID. La figura 2.19. muestra las relaciones existentes entre empleado, factura y detalle_factura.

Figura 2.19. Arrastrar llave primaria



La llave primaria se ha extendido desde empleado hasta detalle_factura. Este caso la entidad detalle_factura posee una llave primaria triple compuesta de los campos número de la entidad factura y de carnet de la entidad empleado. La

Figura 82. Páginas 17 – 20 Libro “Bases de datos principiantes”

entidad factura hereda de empleado su llave primaria (carnet). Esto hace que la entidad factura tenga una llave primaria doble con los campos factura(numero) y empleado(carnet). La entidad detalle_factura hereda de la entidad factura su llave primaria. La entidad factura posee una llave doble, por esta razón detalle_factura posee una llave primaria triple formada por detalle_factura(linea), factura(numero) y empleado(carnet). El diseño de estas relaciones no permite ver el campo que se hereda a las entidades. La única forma que se sabe que este campo existe es por la representación UID en el extremo de la relación.

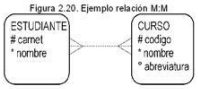
2.2.2.7. Eliminar Relación Muchos a Muchos

Las relaciones de M:M surgen en un principio pero estas deben ser eliminadas y sustituidas por relaciones de 1:M.

Ejemplo 2.8.

Se posee la siguiente relación entre la entidad estudiantes y la entidad curso. La figura 2.20. representa esta relación. Un estudiante se puede asignar varios cursos y un curso puede ser asignado por varios estudiantes.

Figura 2.20. Ejemplo relación M:M



Esta relación se elimina añadiendo una nueva entidad entre la entidad estudiante y la entidad curso. Estas deben ser relacionadas 1:M con la entidad nueva.

La Figura 2.5. representa la relación de asignación. Esta figura presenta como debería de quedar la relación entre la entidad estudiante y la entidad curso después de eliminar la relación M:M.

La entidad asignación se relaciona a la entidad estudiante y a la entidad curso. Esto permite indicar que un estudiante puede asignarse varias veces un curso y un curso puede ser asignado varias veces por un estudiante.

La eliminación de relaciones de M:M algunas veces conlleva a la utilización de relaciones 1:M del tipo UID. Esto quiere decir que la llave primaria de la entidad en la parte de uno, formará parte de la entidad que se crea al eliminar la relación M:M. Esto se realizará por cada entidad que se relacione de M:M. Esta eliminación permite mantener la integridad de los datos en el modelo.

La importancia de eliminar relaciones de M:M radica en que al momento de la implementación se crean una serie de problemas. Esto quiere decir que se tendrán llaves foráneas duplicadas en ambas entidades relacionadas y esto conlleva un alto grado de redundancia de los datos.

2.2.2.8. Relaciones Redundantes

Las relaciones redundantes son aquellas relaciones que permiten que una llave primaria de una entidad A se encuentre como foránea en la entidad B y la llave primaria de la entidad B se regrese como foránea a la entidad A. La otra forma de ver las relaciones redundantes es como posibles caminos que permiten llegar de varias maneras hacia una misma entidad. Estos caminos son caminos redundantes.

Las relaciones redundantes no se deben de permitir, generalmente son eliminadas pero existen casos muy extremos donde podrían ser justificado su uso.

Ejemplo 2.9.

Las entidades estudiante, asignación y curso existen para poder llevar el control de que los estudiantes pueden asignarse varios cursos. La figura 2.21. presenta la forma de ver esta relación.

Figura 2.21. Relación redundante

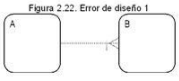
El problema que existe en esta relaciones es que existe una relación directa de M:M entre estudiante y curso. Esta relación se observa que cumple la función de la entidad asignación. La relación de M:M de permitirse generaría redundancia de los datos debido a que este registro ya se encuentra siendo almacenado por la entidad asignación. La relación de M:M generará una nueva copia de las llaves de estudiante y de curso y esto generaría redundancia en el modelo. La relación de M:M debe desaparecer del modelo y de esta forma evitar la redundancia, lo mismo ocurriría si la relación existente entre estudiante y curso fuera de 1:1 y de 1:M. Esto provoca redundancia en los datos del modelo.

2.2.2.9. Errores de Diseño

Cuando se realiza el diseño de un diagrama entidad relación pueden ocurrir errores de diseño. A continuación se listan algunos errores muy frecuentes.

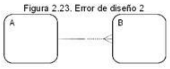
La relación no puede ser opcional del lado de muchos. Este error se encuentra representado en la Figura 2.22. Este error ocurre debido a que la relación de muchos no puede ser opcional. Esto es porque para que una tupla en la entidad A no necesariamente posee una relación con la entidad B. La tupla en la entidad B existe debido a que se encuentra relacionada con alguna tupla de la entidad A. Esta tupla en B si no se encuentra relacionada con la tupla en A no debe de existir. Esto conlleva a indicara que del lado de muchos debe ser obligatorio.

Figura 2.22. Error de diseño 1



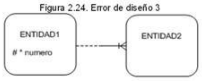
La barra se coloca siempre en el extremo de muchos. Este error se encuentra representado en la Figura 2.23. debido a que la barra se encuentra en el lado de uno. La barra permite indicar que la llave primaria de la entidad A formará parte de la entidad B. Esto no puede determinarse debido a que del lado de uno no existe llave foránea. Esto cambia del lado de muchos debido a que en esta entidad si existe llave foránea referente a la entidad A.

Figura 2.23. Error de diseño 2



La entidad2 no puede existir solamente con el campo número. La entidad2 debe ser eliminada debido a que no provee ninguna funcionalidad en el modelo. Esto quiere decir que la función de llevar el control del campo número se encuentra dado en la entidad1 y por lo tanto no tiene ningún sentido duplicar estos datos en la entidad 2. La aceptación de la entidad2 sería si ésta estuviera formada por más atributos que brinden y añadan información que no puede se puede saber utilizando solamente la entidad2. La figura 2.24. muestra este error de diseño.

Figura 2.24. Error de diseño 3



Las entidades no deben poseer un ciclo referencial. Esto quiere decir que la entidad1 pasará su llave foránea hacia la entidad2, la entidad2 pasará su llave foránea a la entidad3 y la entidad3 pasará su llave foránea a la entidad1. Esto se complica en el caso que además de que se pase la llave foránea se envíe esta como parte de la llave primaria de las otras entidades. Esto haría heredar a las entidades la llave primaria y se obtendría un ciclo infinito. La figura 2.25. muestra este error de diseño.

Figura 2.25. Error de diseño 4

Figura 83. Páginas 21 – 24 Libro “Bases de datos principiantes”

2.2.2.10. Modelado de Estructuras Complejas

Las estructuras complejas se encuentran integradas por los supertipos y subtipos, los arcos, las relaciones recursivas y el modelado en el tiempo.

2.2.2.10.1. Supertipos y Subtipos

Un supertipo es una entidad que se encuentra dividida en grupos más pequeños y mutuamente exclusivos. Un supertipo puede tener atributos propios o pueden ser solamente usados como el nombre de un grupo.

Un subtipo es una entidad que representa un grupo descompuesto dentro del supertipo. Los subtipos poseen atributos propios como también pueden heredar los atributos asignados al supertipo.

Los supertipos y subtipos se pueden utilizar:

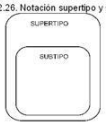
- Cuando un grupo de entidades tienen propiedades especiales que solo existen para éste.
- Cuando hay cierto tipo de funcionalidad que solo aplica a un grupo específico.

Las consideraciones que se deben tener en cuenta para el modelado de supertipo y subtipos son:

- Las entidades de subtipos deben ser mutuamente exclusivas y por consiguiente una instancia de un subtipo no puede también ser una instancia de otro.
- Los atributos y relaciones que son comunes a todos los subtipos son definidas en el nivel del supertipo. Una entidad de subtipo implícitamente hereda todos los atributos, relaciones y funciones del negocio de los supertipos.
- Los subtipos pueden tener atributos y relaciones propias. La opcionalidad de estos atributos y relaciones debe ser reconsiderada.
- Los UID asignados a el supertipo también son asignados al subtipo.

La figura 2.26. muestra la forma como se representa un supertipo que contiene un subtipo.

Figura 2.26. Notación supertipo y subtipo



Ejemplo 2.10.

Una persona es alguien que puede desempeñar distintos papeles. Estos papeles se encuentran dados dependiendo de sus actividades. El entorno estudiantil restringe que una persona puede llegar a ser un estudiante así como puede ser un catedrático.

La entidad persona se puede modelar como si fuera un supertipo. Las entidades estudiante y catedrático pueden ser modeladas como subtipos de la entidad persona.

La entidad persona posee dos atributos los cuales son carnet y nombre. Esto quiere decir que todas las entidades dentro de persona poseerán dos propiedades y tendrán como llave primaria el campo carnet.

La entidad catedrático poseerá cuatro atributos que son carnet, nombre, salario y teléfono.

La entidad estudiante poseerá cuatro atributos que son carnet, nombre, dirección y teléfono. El modelo no reflejará gráficamente los atributos heredados del supertipo hacia los subtipos dentro del subtipo.

La figura 2.27. muestra como quedaría el modelado de la entidad persona y de las entidades catedrático y estudiante.

Figura 2.27. Supertipo y subtipo de persona

Los subtipos se pueden anidar. Esto quiere decir que un subtipo puede contener uno o más subtipos y estos a su vez contener a mas subtipos y así sucesivamente. El nivel que se recomienda para anidar subtipos es de dos niveles. El ser mayor de dos niveles vuelve muy difícil el comprender el modelo. Las reglas y consideraciones para supertipos y subtipos siguen siendo las mismas, incluso al seguir anidando las entidades.

Ejemplo 2.11.

El ejemplo 2.11 se basa en el ejemplo 2.10. Este ejemplo se ha extendido de tal forma que pueda albergar subtipos anidados. Un estudiante dentro del ciclo académico puede ser de dos tipos. El estudiante puede ser estudiante regular ó estudiante de maestría. La entidad estudiante entonces puede anidar otras dos entidades que corresponden a regular y a maestría. La entidad regular posee el atributo tutor y la entidad maestría posee el atributo que corresponde al número de registro.

La figura 2.28 muestra la forma como quedaría la entidad persona junto con sus subtipos. El subtipo anidado (maestría) contiene una llave primaria propia, por lo tanto la llave primaria de la entidad maestría es una llave primaria doble, conformada por carnet y no_registro.

Figura 2.28. Subtipo persona anidado

2.2.2.10.2. Arcos

Los arcos son utilizados debido a que algunas veces al construir el modelo, se encuentran entidades que poseen relación con entidades A y B. Estas relaciones son válidas pero nunca al mismo tiempo. El tipo de relación es conocido como una relación exclusiva. Estas relaciones se modelan por medio de un arco.

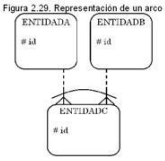
Los arcos poseen una serie de reglas para su adecuada construcción. Estas reglas son:

- Todas las relaciones que finalizan en el arco deben tener la misma opcionalidad, todas opcionales o todas obligatorias.
- Una relación puede estar sólo en un arco.
- Un arco siempre pertenece a una entidad.
- Las relaciones que finalizan deben venir de la misma entidad.
- No hay límite en el número de relaciones que se pueden poner en un arco.
- Las relaciones dentro de un arco a menudo tienen el mismo nombre.

La mayor parte del tiempo se recomienda que se utilice diagramar supertipos en lugar de arcos. Una serie de convenciones han sido acordadas para modelar arcos. El arco se dibuja cruzando todas las relaciones que se deseen incluir. El arco se cierra hacia la entidad que lo está utilizando. La figura 2.29. muestra la forma de cómo representar un arco. El arco pertenece a la entidad c.

Figura 84. Páginas 25 – 28 Libro “Bases de datos principiantes”

Figura 2.29. Representación de un arco



Ejemplo 2.12.

Una empresa maneja sus ventas de tal forma que sus clientes pueden ser considerados como personas individuales o empresas. La venta es realizada únicamente para una persona o para una empresa, pero no ambas. Esto se puede modelar utilizando un arco. La entidad que contendrá al arco será la entidad venta. Las entidades persona y empresa serán las que serán utilizadas una vez para cada venta. Los atributos de la entidad persona son código, cedula, nombre. Los atributos de la entidad empresa son código, nombre y número de empleados. Los atributos de la entidad venta son transacción y el nombre del vendedor. La figura 2.30. muestra la representación gráfica del arco de ventas.

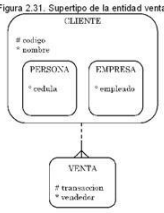
Figura 2.30. Arco de la entidad ventas

Ejemplo 2.13.

A partir del ejemplo 2.12. se desea realizar el modelo con la diferencia de que en lugar de utilizar un arco, se utilicen supertipos y subtipos. El supertipo

comprende las entidades persona y empresa. Este supertipo se considera como cliente. La relación que en la figura 2.30. parte de persona y empresa hacia venta, en este caso partirá del supertipo cliente hacia la entidad venta. El supertipo cliente contendrá los atributos que se encuentran en empresa y cliente. Estos son el código y el nombre. Los atributos propios de empresa y cliente quedarán dentro de cada subtipo. La figura 2.31. muestra la representación mediante supertipos y subtipos del ejemplo 2.12.

Figura 2.31. Supertipo de la entidad venta



2.2.2.10.3. Relaciones Recursivas

Los conceptos de divergencia y convergencia surgen al momento de realizar modelos de datos.

La convergencia es el proceso por el cual se simplifica un modelo sin la perdida de definición. Esto quiere decir que se puede reducir el número de entidades y relaciones, pero se corre el riesgo de omitir significativamente las reglas del negocio. La convergencia permite la creación de estructuras recursivas. Las estructuras recursivas no son más que entidades que se relacionan a si mismas. Estas entidades se relacionan de forma 1:M ó de M:M y son totalmente opcionales. Esto es debido a que de no ser así las consultas se convierten en un ciclo infinito.

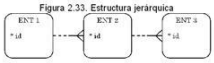
Las entidades recursivas se utilizan al momento en que una llave primaria se arrastra más de cinco niveles. Estas entidades también son utilizadas cuando la

granularidad de la relación puede llegar a ser infinita. La figura 2.32. representa la forma gráfica de ver una estructura recursiva.

Figura 2.32. Estructura recursiva

La divergencia es lo opuesto de la convergencia. Esto quiere decir que se añade más detalle en el modelo, pero se corre el riesgo de obtener niveles de innecesaria complejidad. La divergencia da lugar a realizar estructuras jerárquicas. Estas estructuras son entidades que se encuentran relacionadas de 1:M con o sin UID. Las estructuras jerárquicas poseen un limite de niveles. Los modelos pueden llegar a poseer cinco niveles o menos. La figura 2.33. representa una estructura jerárquica.

Figura 2.33. Estructura jerárquica



Ejemplo 2.14.

La institución geográfica internacional desea elaborar un modelo de datos capaz de albergar información de las regiones del mundo. Esto quiere decir que el modelo debe ser capaz de guardar información del área a la que pertenece un pais. Un pais posee una o más regiones. Una región que se encuentra en un pais comprende uno o más departamentos. Estos departamentos geográficamente se encuentran integrados por municipios. A partir de esto se desea que el modelo se estructure jerárquicamente y recursivamente. Esto para poder comparar posteriormente la complejidad de ambos modelos y optar por el modelo que más se adapte a las necesidades de la institución.

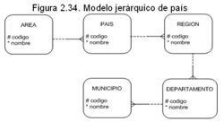
El primer modelo que se realizará será el que utiliza estructura jerárquica. La jerarquía se encuentra compuesta por:

- Area
  - Pais
    - Region
      - Departamento

  - Municipio

Las entidades que componen la jerarquía se encuentran relacionadas de 1:M, donde del lado de uno son opcionales y de lado de muchos son obligatorias. Los atributos de todas las entidades son código y nombre. El nivel de la jerarquía es de cinco niveles. Por ejemplo la jerarquía permite tener datos que se llamen igual pero geográficamente se diferencian por lo cual los datos no podrían tratarse como el mismo. Esto quiere decir que dos municipios que se llamen San Pedro pertenecen cada uno a departamentos distintos, por lo cual el dato de San Pedro debe encontrarse dos veces en la entidad municipio y estar relacionado a distinto departamento. La figura 2.34. muestra se muestra como quedaría el modelo jerárquico.

Figura 2.34. Modelo jerárquico de país



El segundo modelo que se realizará será el de estructura recursiva. Este modelo consta de una sola entidad. La entidad debe poseer un nombre genérico que englobe a todas las demás entidades. El caso es que se refiere a lugares como pais, municipios y departamentos. Esto permite que el nombre de la entidad sea dado como lugar. La entidad lugar debe poseer todos los atributos de todas las entidades que se encuentran relacionadas jerárquicamente. Estos atributos son código y nombre. La entidad lugar además posee un atributo extra que se denomina tipo. Este atributo permite indicar si el dato en la entidad es un área, un pais, una región, un departamento o un municipio. Es importante hacer notar que si alguna o más entidades dentro de la jerarquía poseen atributos propios, estas pueden ser consideradas en la entidad recursiva. Los atributos propios tendrían que ser opcionales para que de esta forma si alguna entidad que no los posea puede dejarlos como nulos y así respetar la integridad. La figura 2.35. muestra el modelo recursivo.

Figura 85. Páginas 29 – 32 Libro “Bases de datos principiantes”

Figura 2.35. Modelo recursivo de pais

2.2.2.10.4. Roles

La palabra rol se refiere al papel que desempeña determinada persona en una empresa o asociación. Los roles pueden ser utilizados en el modelado de datos. Estos permiten que ciertas entidades desempeñen un papel al ser asociadas con otra entidad. Los roles en ocasiones se manejan como entidades separadas. Esto porque a veces se tienen campos que son obligatorios en algunos roles y otras veces existen relaciones específicas entre los roles.

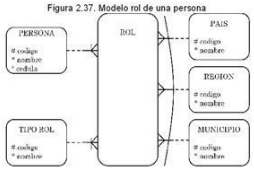
Ejemplo 2.15.

Una persona puede desempeñar distinto cargo dependiendo del lugar donde trabaja. Esto quiere decir que si una persona trabaja dirigiendo un país, ésta es el presidente del mismo. Otra persona puede trabajar dirigiendo una región de un país y desempeña el papel de gobernador. Otra persona puede trabajar dirigiendo un municipio y esta desempeña el papel de alcalde del municipio. La figura 2.36. el modelo que muestra lo anterior.

Figura 2.36. Modelo papeles de una persona por lugar

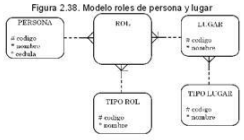
Las entidades presidente, gobernador y alcalde pueden ser comprendidas como una entidad rol. Esta entidad maneja los tres papeles que desempeña una persona. La entidad rol dependerá de otra entidad que será denominada como tipo rol. Esta entidad permite guardar los roles es decir presidente, gobernador y alcalde. La entidad rol será como una entidad donde se asignará una persona a un tipo de rol y a un país, región o municipio. La figura 2.37. muestra esta entidad rol que albergará las funciones de la entidad persona dependiendo de las entidades país, región y municipio. Estas tres entidades solamente pasarán su llave foránea una y solamente una. Esto permite que se añada un arco para la entidad rol y este sirve para restringir la entidad país, región y municipio.

Figura 2.37. Modelo rol de una persona



El modelo está bien al utilizar la entidad rol y la entidad tipo rol. El arco puede ser sustituido por otra entidad rol. Esta entidad guardará los papeles que pueden ser utilizados por los lugares, en este caso país, región y municipio. La entidad se nombrará lugar debido a que estas entidades corresponden a lugares y la entidad tipo se referirá a tipos de lugares. La entidad tipo lugar definirá si se trata de un país, un municipio o una región. La entidad lugar en cambio definirá lugares concretos. Esto quiere decir que guardará los datos propios de un país, de una región o de un municipio. La figura 2.38. muestra el modelo añadiendo una entidad lugar y tipo lugar.

Figura 2.38. Modelo roles de persona y lugar



2.2.2.10.5. Modelado en el Tiempo

El modelado en el tiempo surge de la necesidad de poder acomodar los modelos entidad relación para que puedan llevar datos históricos.

El llevar el registro histórico necesita crear una nueva entidad. Esta nueva entidad servirá para poder llevar el control de los datos históricos. Las entidades que se relacionan con esta entidad tendrán que pasar sus llaves foráneas y éstas formarán parte de la llave primaria de la nueva entidad. Los atributos que tendrán esta entidad son un atributo de fecha de inicio el cual es totalmente obligatorio y además forma parte de la llave primaria. La fecha de fin la cual será un atributo opcional debido a que éste puede en determinado momento no existir lo que indicará que el registro se encuentra activo en el caso de que se trate de una persona trabajando dentro de una empresa. Este atributo de poseer un valor puede significar que un empleado ha dejado de laborar en esa fecha o que ha sido cambiado de algún departamento.

Los requerimientos deben ser validados para poder almacenar los datos históricos. Esto quiere decir que debe ser importante al negocio llevar estos datos porque de lo contrario guardar información histórica puede ser demasiado costoso.

Las tres preguntas que pueden permitir contribuir a decidir llevar un registro histórico son:

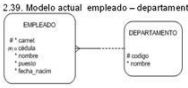
- ¿Una auditoria es requerida?
- ¿Los valores de los atributos pueden cambiar con el tiempo?
  - ¿Pueden las relaciones cambiar con el tiempo?
  - ¿Se necesita realizar una consulta de datos viejos?

- ¿Se necesita guardar versiones viejas?

Ejemplo 2.16.

Un propietario de una empresa quiere llevar el historial de los departamentos en los que han trabajado sus empleados. El modelo actual le resulta insuficiente para poder obtener esta información. La figura 2.39. muestra el modelo actual que utiliza esta persona para llevar sus registros.

Figura 2.39. Modelo actual empleado – departamento



El modelado en el tiempo necesita que se cree una nueva entidad entre las entidades empleado y departamento. Esta nueva entidad se nombrará como asignación. La entidad asignación poseerá dos fechas. Estas fechas corresponden a la fecha de inicio donde el empleado empieza a trabajar en ese departamento y la fecha de fin donde el empleado deja de trabajar en ese departamento. El atributo de la fecha de inicio forma parte de la llave primaria. Las llaves foráneas de empleado y de departamento completan la llave primaria de la entidad asignación. La figura 2.40. muestra el nuevo modelo que permite llevar los registros históricos de los empleados y departamentos.

Figura 2.40. Modelo asignación de empleados y departamentos

2.2.2.10.6. Trampas de Conexión

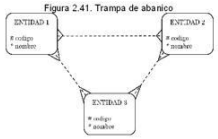
Una trampa de conexión ocurre cuando un modelo entidad relación se encuentra abierto a una mala representación. El término trampa es usado debido a que el modelo puede conducir a que dos objetos aparezcan

Figura 86. Páginas 33 – 36 Libro “Bases de datos principiantes”

conectados en el diagrama, esto por consiguiente infiere especifica información de este.

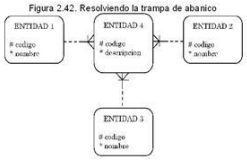
Un caso específico de una trampa de conexión se conoce como trampa de abanico. Esta trampa surge debido a relaciones complejas entre tres o más entidades. Una trampa de abanico forma un anillo de relaciones de M:M o entidades entrecruzadas. Este anillo se conoce como el anillo de la muerte. La figura 2.41. muestra una trampa de abanico.

Figura 2.41. Trampa de abanico



Las trampas de abanico se eliminan al crear una relación entre las entidades que se relacionan de M:M. (Es importante denotar que solo se puede realizar esta solución si todas las entidades se encuentran relacionadas de M:M). Esta nueva entidad recibe las llaves foráneas como primarias y la relación se encuentra dada por 1:M. La entidad también puede poseer sus propios atributos así como atributos llave primaria que serán utilizados con las llaves primarias que hereda. La figura 2.42. muestra la solución de la trampa de abanico.

Figura 2.42. Resolviendo la trampa de abanico



Ejemplo 2.17.

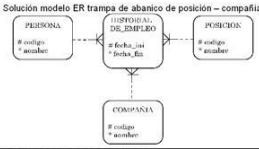
Diseñar un modelo entidad relación para llevar el historial de los empleados. El modelo debe ser capaz de llevar la posición alcanzada por la persona, la compañía en la que trabajó y las fechas que esa persona mantuvo la posición. Una persona puede tener una posición especifica dentro de la misma compañía múltiples veces durante su carrera.

La solución de este problema conduce a una trampa de abanico. Esto es debido a que la fecha de la posición parece ser un atributo de relaciones M:M dentro de las entidades persona, posición y compañía. El modelo en este momento sería capaz de indicar las fechas que una posición existe dentro de una compañía, las fechas que una persona trabajó para una compañía y las fechas que una persona mantuvo una posición. La figura 2.43. muestra este modelo.

Figura 2.43. Modelo ER trampa de abanico de posición – compañía – empleado

La solución de esta trampa de abanico permitirá crear una entidad que se llamará historial de empleo. Esta entidad poseerá dos fechas. La fecha de inicio que será llave primaria también y la fecha de fin. La figura 2.44. muestra este modelo.

Figura 2.44. Solución modelo ER trampa de abanico de posición – compañía – empleado



2.3. Diseño de Base de Datos

2.3.1. Mapeo Conceptual

El mapeo conceptual se realiza cuando se posee un diagrama o modelo de datos finalizado. El mapeo conceptual es un paso previo a la implementación real de un modelo de datos.

El mapeo conceptual se caracteriza por ser independiente del DBMS. Esta independencia lo vuelve en cierto grado genérico.

El mapeo conceptual es importante que se realice antes de que se realice el esquema conceptual. La Tabla 2.1. es la representación lógica y física de cómo se observan las entidades.

Tabla 2.1. Representación lógica y física

Lógica	Física
Entidades	Tablas
Atributos ó propiedades	Campos
Tuplas	Registros
PK	Restricciones (*Constraints*)
FK	

El mapeo conceptual se realiza por medio de definir una tabla con cuatro filas. Esta tabla se realiza una vez por cada entidad que contenga el diagrama o modelo de datos. Las filas de esta tabla representan:

- Columna: esta representa el nombre del atributo.
- Tipo de llave: representa si el atributo es llave única (UK), llave primaria (PK) ó llave foránea (FK). Esta fila pude ser vacía si el atributo no es ninguna de las llaves indicadas anteriormente.
- Opcionalidad: se coloca un *NULL* (°) o *NOT NULL* (*) dependiendo de si el atributo es opcional u obligatorio.
- Datos de Entrada: se recomienda colocar 3 ejemplos para cada atributo. Lo importante de esta columna es que aunque se poseen datos de muestra, no se especifica en ningún momento el tipo del campo y esto es debido a que es independiente del DBMS.

El número de columnas está dado por cada atributo que posea la entidad. En caso se poseer llaves foráneas se añade una columna por cada llave foránea que posea la entidad.

Ejemplo 2.18.

A partir de la figura 2.5. se desea mapear cada una de las entidades que intervienen en el modelo. Las entidades son la entidad estudiante, la entidad curso y la entidad asignación.

La entidad estudiante no posee llaves foráneas. La llave primaria de esta entidad es el carnet del estudiante. Esta entidad también posee los atributos cédula, nombre, teléfono y fecha de nacimiento. La entidad estudiante al ser mapeada corresponde a la Tabla 2.2.

Tabla 2.2. Mapeo de entidad estudiante

Columna	Carnet	cedula	nombre	telefono	fecha_nac
Opcionalidad	NOT NULL	NULL	NOT NULL	NOT NULL	NOT NULL
Columna	PK				
Dato de Muestra	2003123	A-1 32	Juan	12344	11/02/1985
	2005123	U-1 23	Pedro	12345	11/02/1987
	2004123	A-1 21	José	12347	11/02/1986

La entidad curso no posee llaves foráneas. La llave primaria de esta entidad es el código del curso. Esta entidad también posee los atributos nombre y descripción. La entidad curso al ser mapeada corresponde a la Tabla 2.3.

Tabla 2.3. Mapeo de entidad curso

Columna	código	Nombre	descripcion
Opcionalidad	NOT NULL	NOT NULL	NOT NULL
Columna	PK		
Dato de Muestra	0774	BD1	Base de datos

Figura 87. Páginas 37 – 40 Libro “Bases de datos principiantes”

	0772	IPC2	Introducción y programación de computadoras 2
	0771	IPC1	Introducción y programación de computadoras 1

La entidad asignación posee dos llaves foráneas. Estas llaves corresponden al carnet de la entidad estudiante y al código de la entidad curso. Los atributos propios de la entidad asignación solamente es descripción. La entidad asignación al ser mapeadas quedaria conforme a la Tabla 2.4. Esta tabla presenta el mapeo conceptual de la entidad asignación.

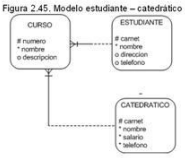
Tabla 2.4. Tabla de mapeo de la entidad asignación

Columna	no asignación	Estado	semestre	carnet estudiante	codigo curso
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL
Columna	PK			FK2	FK1
Dato de Muestra	1	A	200801	200312942	0771
	2	A	200802	200312942	0772
	3	A	200801	200312954	0771

Ejemplo 2.19.

La figura 2.45. muestra un modelo correspondiente a la relaciones que existen entre un estudiante, un catedrático y un curso. El mapeo correspondiente a estas entidades no varia mucho del mapeo de las entidades del ejemplo 2.18.

Figura 2.45. Modelo estudiante – catedrático



La entidad estudiante no posee llaves foráneas. La llave primaria de esta entidad es el carnet del estudiante. Esta entidad también posee los atributos nombre, dirección y teléfono. La entidad estudiante al ser mapeada corresponde a la Tabla 2.5.

Tabla 2.5. Mapeo de entidad estudiante

Columna	Carnet	nombre	Dirección	telefono
Opcionalidad	NOT NULL	NOT NULL	NULL	NULL
Columna	PK			
Dato de Muestra	2003123	Juan	11 calle 7-61 zona 1	12344
	2005123	Pedro	11 calle 7-61 zona 2	12345
	2004123	José	11 calle 7-61 zona 3	12347

La entidad catedrático no posee llaves foráneas. La llave primaria de esta entidad es el carnet del catedrático. Esta entidad también posee los atributos nombre, salario y teléfono. La entidad catedrático al ser mapeada corresponde a la Tabla 2.6.

Tabla 2.6. Mapeo de entidad catedrático

Columna	Carnet	nombre	direccion	telefono
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL

Columna	PK				
Dato de Muestra	0001123	Luis	2000.20	12344	
	0002123	Fernando	3000.00	12345	
	0003123	Marlon	3000.00	12347	

La entidad curso posee dos llaves foráneas y estas llaves primarias forman parte de la llave primaria de esta entidad. Estas llaves corresponden al carnet de la entidad estudiante y al carnet de la entidad catedrático. Los atributos propios de la entidad curso son nombre y descripción. La entidad curso al ser mapeadas quedaria conforme a la Tabla 2.7. Esta tabla presenta el mapeo conceptual de la entidad curso.

Tabla 2.7. Tabla de mapeo de la entidad curso

Columna	numero	nombre	descripción	carnet estudiante	carnet catedratico
Opcionalidad	NOT NULL	NOT NULL	NULL	NOT NULL	NOT NULL
Columna	PK			FK1, PK	FK2, PK
Dato de Muestra	1	BD1	Bases de datos 1	200312942	0001123
	2	IPC2		200312942	0002123
	3	IPC1		200312954	0003123

2.3.1.1. Mapeando Entidades que Arrastran la Llave Primaria

El arrastrar la llave primaria en el modelo solamente se refleja hasta el momento en que se desea mapear la entidad. Esto quiere decir que en el modelo el simbolo de UID es el que indica que una llave foránea se convierte en parte de la llave primaria, pero este atributo no se aprecia formalmente en la entidad. El mapeo de la entidad es donde este UID se convierte en un campo. La sección 2.2.2.6. muestra lo referente a la representación en el modelo de este tipo de casos. El ejemplo 2.7. trata sobre dos entidades que reciben llaves foráneas y que estas llaves foráneas se convierten en parte de la llave primaria

de cada entidad. La figura 2.19. muestra el modelo de este caso. El mapeo para las entidades se realizará de forma individual.

El modelo se encuentra integrado por tres entidades. Estas entidades son: empleado, factura y detalle_factura.

La entidad empleado se mapea fácilmente debido a que no posee ninguna llave foránea. La figura 2.8. muestra el mapeo de la entidad empleado.

Tabla 2.8. Mapeo de empleado

Columna	Carnet	cedula	Nombre	Puesto	fecha_nac
Opcionalidad	NOT NULL	NULL	NOT NULL	NOT NULL	NOT NULL
Columna	PK				
Dato de Muestra	2003123	A-1 234	Mario	Supervisor	11/21/1980
	2005123	A-1 244	José	Operador	11/21/1982
	2004123	U-1 344	Felipe	Operador	11/21/1983

La entidad factura es un poco más complicada de mapear debido a que posee una llave foránea que viene de la entidad empleado. Esta llave foránea forma parte de la llave primaria de la entidad factura. Los atributos de esta entidad son entonces numero, fecha, cliente y carnet del empleado. La tabla 2.9. muestra el mapeo de esta entidad.

Tabla 2.9. Mapeo de factura

Columna	carnet empleado	numero	Fecha	cliente
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL
Columna	FK1, PK	PK		
Dato de Muestra	2003123	123	03/30/2009	Juan Pérez
	2005123	125	03/30/2009	José Pérez
	2004123	126	03/30/2009	Pedro Pérez

Figura 88. Páginas 41 – 44 Libro “Bases de datos principiantes”

El mapeo de la entidad detalle_factura se complica un poco debido a que este adquiere por llave foránea dos atributos que vienen de la misma entidad y además forman parte de la llave primaria de esta entidad. La llave primaria compuesta de la entidad detalle_factura es linea, numero factura y carnet de empleado. Los demás atributos de la entidad son producto y cantidad. La tabla 2.10. muestra el mapeo de la entidad detalle_factura.

Tabla 2.10. Mapeo de la entidad detalle_factura

Columna	carnet	numero	linea	producto	Cantidad
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL
Columna	FK1, PK	FK1, PK	PK		
Dato de Muestra	2003123	123	1	Televisión tipo 1	1
	2005123	123	2	Televisión tipo 2	2
	2004123	125	1	Estufa tipo 1	1

2.3.1.2. Mapeo de Supertipos y Subtipos

El mapeo de los supertipos puede ser de dos formas.

La primera forma considera al supertipo como una sola entidad que contiene a los subtipos. Esto quiere decir que los atributos de cada subtipo pasan a ser propios de la entidad supertipo y son completamente opcionales debido a que estos atributos no corresponderán a un dato de un subtipo pero no a ambos. Las relaciones propias de cada subtipo serán heredadas por el supertipo.

La segunda forma considera a los supertipos como entidades separadas. Esto quiere decir que se realizará una entidad por cada subtipo. Los atributos propios del supertipo se heredan al igual que las relaciones a las entidades de los subtipos y mantienen su opcionalidad. La figura 2.46. servirá para poder ejemplificar las dos formas de cómo mapear supertipos y subtipos.

Ejemplo 2.20.

La figura 2.27. modela un supertipo persona que contiene dos subtipos un subtipo corresponde a catedrático y el otro subtipo corresponde a estudiante.

El mapeo de la primera forma de este supertipo corresponde crear una sola entidad. Los atributos de las entidades catedrático y estudiante quedan como opcionales en la entidad supertipo. La tabla 2.11. muestra la forma de mapear un supertipo utilizando una única tabla.

Tabla 2.11. Mapeo supertipo persona forma 1

Columna	carnet	nombre	salario	telefono1	direccion	teléfono2
Opcionalidad	NOT NULL	NOT NULL	NULL	NULL	NULL	NULL
Columna	PK					
Dato de Muestra	200312	Mario	123.42	123456		
	200412	José			2a calle 9-43 z. 2	234569
	200512	Juan	111.20	123458		

La segunda forma de mapear el supertipo de la figura 2.27. corresponde a realizar dos tablas. Estas tablas conservarán el nombre de los subtipos y heredaran la opcionalidad de los atributos del supertipo persona. Los atributos propios de cada subtipo mantienen su opcionalidad. La tabla 2.12. muestra el subtipo catedrático.

Figura 2.12. Mapeo subtipo catedrático forma 2

Columna	carnet	Nombre	salario	Teléfono
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL
Columna	PK			
Dato de Muestra	200312	Mario	123.42	123456
	200412	José	112.20	123457

	200512	Juan	111.20	123458

La tabla 2.13. muestra el subtipo estudiante. La opcionalidad de los atributos de los subtipos se mantiene.

Figura 2.13. Mapeo subtipo estudiante forma 2

Columna	carnet	Nombre	direccion	Teléfono
Opcionalidad	NOT NULL	NOT NULL	NULL	NULL
Columna	PK			
Dato de Muestra	200312	Mario	1a calle 7-61 z. 1	234568
	200412	José	2a calle 9-43 z. 2	234569
	200512	Juan	10a Av. 9-55 zona 4	234579

La tabla 2.13. muestra el subtipo estudiante, si existieran relaciones para cada subtipo estas relaciones se mantienen. Las relaciones del supertipo serían heredadas para cada subtipo.

Ejemplo 2.22.

El ejemplo 2.22. presenta un ejemplo más complejo de mapear supertipos y subtipos. Este ejemplo se basa en el modelo de la figura 2.28. El modelo consta de un supertipo persona y dos subtipos catedrático y estudiante. El subtipo estudiante posee dos subtipos. Estos son el subtipo regular y el subtipo maestría.

El mapeo se realizará de las dos formas que se han enseñado. La primera forma es realizar una tabla, para este caso será la tabla del supertipo persona. La llave primaria que corresponde al subtipo maestría, pasa a ser un atributo opcional y ya no compone la llave primaria. Esto es debido a que los demás subtipos no pueden poseer este atributo como obligatorio. La tabla 2.14. presenta esta única tabla.

Tabla 2.14. Mapeo supertipo persona

Columna	carnet	nombre	salario	telefono 1	Direccion	teléfono 2	tutor	no registro
Opcionalidad	NOT NULL	NOT NULL	NULL	NULL	NULL	NULL	NULL	NULL
Columna	PK							
Dato de Muestra	200312	Mario	123.42	123456				
	200412	José			2a calle 9-43 z. 2	234569		100398
	200512	Juan			10a Av. 9-55 zona 4	234579	Mateo	

La segunda forma corresponde a mapear una entidad por cada subtipo. Este caso se realizará una tabla para el subtipo catedrático. El subtipo estudiante enfrenta dos opciones de mapeo. La primera opción corresponde a mapear una sola tabla partiendo del subtipo estudiante o mapear dos tablas que corresponden al subtipo regular y al subtipo maestría. El ejemplo realizará dos tablas a partir del subtipo estudiante. El mapeo de la entidad catedrático fue realizado en la tabla 2.12. La tabla 2.15. muestra el mapeo de la entidad estudiante regular.

Figura 89. Páginas 45 – 48 Libro “Bases de datos principiantes”

Tabla 2.15. Mapeo entidad estudiante regular

Columna	carnet	Nombre	direccion	teléfono	tutor
Opcionalidad	NOT NULL	NOT NULL	NULL	NULL	NOT NULL
Columna	PK				
Dato de Muestra	200312	Mario	1a calle 7-71 z. 1	243433	Mateo
	200412	José	2a calle 9-43 z. 2	234569	Pedro
	200512	Juan	10a Av. 9-55 zona 4	234579	Mateo

La llave primaria que corresponde al subtipo maestría será añadida debido a que se eligió mapear tabla por subtipo. La tabla 2.16. muestra el mapeo de la tabla estudiante de maestría.

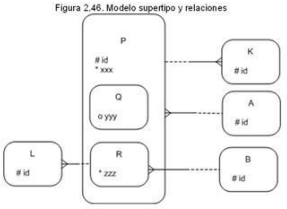
Tabla 2.16. Mapeo entidad estudiante maestria

Columna	carnet	Nombre	Dirección	teléfono	no registro
Opcionalidad	NOT NULL	NOT NULL	NULL	NULL	NOT NULL
Columna	PK				
Dato de Muestra	200312	Mario	11 Av. 9-12 z. 3	234432	100033
	200412	José	2a calle 9-43 z. 2	234569	100398
	200512	Juan	10a Av. 9-55 zona 4	234579	100043

Ejemplo 2.23.

Los ejemplos de mapeo de supertipos vistos hasta este momento no poseen ninguna relación. Este ejemplo se basa en un modelo de supertipos y subtipos que posee relaciones con otras entidades. La figura 2.46. presenta este modelo. Este modelo es solamente de ejemplo y no modela nada del mundo real.

Figura 2.46. Modelo supertipo y relaciones



El modelo de la figura 2.46. posee cuatro entidades externas al supertipo, de estas entidades dos brindan llaves foráneas y dos reciben llaves foráneas del supertipo y los subtipos. El mapeo iniciará realizando la forma uno que consiste en realizar una sola tabla tomando el supertipo. La tabla P adquiere la llave foránea proveniente de la entidad A y su opcionalidad se respeta. La tabla P también adquiere una llave foránea que proviene de la entidad B hacia el subtipo R que se encuentra dentro de la tabla P. La opcionalidad de esta llave foránea debe ser opcional debido a que estamos realizando una única tabla a partir del supertipo P, y esta llave foránea no siempre existirá si se están guardando datos del tipo de la entidad Q. El atributo obligatorio de la entidad R pasa a ser opcional por la misma razón. La tabla 2.17. brinda el mapeo tomando el supertipo como una sola tabla.

Tabla 2.17. Mapeo supertipo P

Columna	id	Xxx	Yyy	zzz	id_A	id_B
Opcionalidad	NOT NULL	NOT NULL	NULL	NULL	NOT NULL	NULL
Columna	PK				FK1	FK2
Dato de Muestra	1	Dato 1	Dato 1		1	
	2	Dato 2		Dato 2	1	1

	3	Dato 3	Dato 3	1	

El mapeo de las entidades A y B se omite debido a que es irrelevante para este ejemplo mapearlas debido a que son entidades simples que en ejemplos anteriores se ha demostrado como realizar su mapeo.

El mapeo de la entidad K se realizará debido a que adquiere una llave del supertipo. Esto sucede también para la entidad L. La tabla 2.18. muestra el mapeo de la entidad K. El atributo id_P es obligatorio debido a que cada dato en K debe estar relacionado a un dato en P.

Tabla 2.18. Mapeo entidad K forma 1

Columna	Id	id_P
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	FK
Dato de Muestra	1	1
	2	1
	3	1

El mapeo de la entidad L se muestra en la tabla 2.19. El atributo id_P es obligatorio debido a que cada dato en L debe estar relacionado a un dato en P siempre y cuando este dato pertenezca al tipo R y no a Q.

Tabla 2.19. Mapeo entidad L forma 1

Columna	Id	id_P
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	FK
Dato de Muestra	1	2
	2	2
	3	2

El modelo de la figura 2.46. puede ser mapeado tomando cada uno de los subtipos como tablas independientes. Esto permite tener una tabla para el subtipo Q y otra tabla para el subtipo R.

La entidad Q hereda los atributos del supertipo P y las relaciones que se relacionan con este supertipo. La entidad Q no recibe ninguna relación directa por lo tanto solamente hereda las relaciones del supertipo. La opcionalidad de esta relación del supertipo se mantiene. La tabla 2.20. presenta el mapeo del subtipo Q.

Tabla 2.20. Mapeo subtipo Q

Columna	Id	Xxx	Yyy	id_A
Opcionalidad	NOT NULL	NOT NULL	NULL	NOT NULL
Columna	PK		FK	
Dato de Muestra	1	Dato 1	Dato 1	1
	2	Dato 2	Dato 2	1
	3	Dato 3	Dato 3	2

La entidad R hereda los atributos y las relaciones del supertipo P. Esta entidad además posee una relación directa de la entidad B. La tabla 2.21. presenta el mapeo del subtipo R.

Tabla 2.21. Mapeo subtipo R

Columna	Id	xxx	Zzz	id_A	id_B
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL
Columna	PK			FK1	FK2
Dato de Muestra	1	Dato 1	Dato 1	1	1
	2	Dato 2	Dato 2	1	2
	3	Dato 3	Dato 3	2	1

Figura 90. Páginas 49 – 52 Libro “Bases de datos principiantes”

El mapeo de la entidad L es normal y hereda únicamente una llave foránea de la tabla R. La tabla 2.22. presenta el mapeo de la entidad L.

Tabla 2.22. Mapeo entidad L forma 2

Columna	Id	id_R
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	FK
Dato de Muestra	1	1
	2	1
	3	2

El mapeo de la entidad K presenta una dificultad. Esto surge a raiz que la relación pertenecia al supertipo P y esta fue heredada a cada tabla (Q y R). Esto origina que solamente un atributo pueda existir en cada dato de la entidad K. La llave foránea o viene de Q o viene de R pero no puede venir de ambas a la vez. Esto se soluciona utilizando un arco. El mapeo de la entidad K se mostrará en la siguiente sección.

2.3.1.3. Mapeo de Arcos

El mapeo de un arco se centra en la entidad a la que pertenece el arco. El mapeo de arcos al igual que el mapeo de supertipos y subtipos se puede realizar de dos formas.

La primera forma se basa en realizar una tabla con cada atributo propio de la entidad. Luego se añadirá una columna que contendrá la llave foránea que pertenece a cada una de las entidades que componen el arco. La tabla también debe contener otra columna que servirá para indicar el tipo de dato que es la columna de la llave foránea. Esto quiere decir que esta columna tipo permitirá indicar si el atributo de la llave foránea corresponde a la entidad x que forma el arco. La entidad x debe entenderse como cualquier entidad que integra el arco. Esta forma al realizar el esquema conceptual puede producir un error debido a que los DBMS no pueden referenciar para un mismo atributo dos tablas.

La segunda forma consiste en añadir una columna (del tipo opcional) a la tabla a la que pertenece el arco, por cada una de las entidades que integran el arco. Esto provocará que exista n columnas donde n es la cantidad de entidades que componen el arco. Esta forma funciona correctamente para la mayoria de los casos, siempre y cuando las relaciones que integran el arco no formen parte de la llave primaria. Esto debido a que las columnas deben ser opcionales y si son parte de la llave primaria esto contradice que una llave primaria no puede ser nula. Este problema se puede evitar si se declaran obligatorias estas columnas pero debe existir un registro en la entidad a la que hace referencia para que se pueda referenciar. Por ejemplo un arco se encuentra formado por la entidad A y la entidad B y el arco pertenece a una entidad C. La entidad A y B se relacionan de tal forma que la llave foránea de A y de B forman parte de la llave primaria de la entidad C. El mapeo del arco se realiza de la segunda forma. Esto indica que la entidad C posee dos columnas obligatorias, en la cual una referencia hacia A y la otra hacia B. Esto obliga a que dentro de A exista un registro que permite indicar que la referencia es nula. Esto también ocurre para B, entonces si C referencia a un registro de A, C referenciará en este mismo momento al registro de referencia nula de B. La llave primaria mantendría su integridad debido a que en ningún momento esta obtendrá un atributo nulo. El registro de referencia nula de B es el registro que se llama nulo o ninguno y que solamente existe en A y en B para poder mantener la integridad de la llave primaria de C.

Ejemplo 2.24.

Este ejemplo será realizado utilizando la primera forma de mapear arcos. El ejemplo se basa en el modelo de la figura 2.30. Esta figura muestra el modelo del arco de una entidad venta que integra el arco con una entidad persona y una entidad empresa. El mapeo de la entidad persona y empresa no implica mayor problema por lo tanto es omitido. El ejemplo se centrará en la entidad que conforma el arco (venta).

El mapeo de la entidad venta posee como llave primaria el atributo transacción y como atributo no nulo el nombre del vendedor. La tabla venta debe contener una columna que contendrá el código de la entidad persona o de la entidad empresa. Esta tabla además tendrá una columna que permitirá indicar el tipo de la llave primaria (si es de la entidad persona o de la entidad empresa). La tabla 2.23. muestra el mapeo de la entidad venta.

Tabla 2.23. Mapeo del arco de la entidad venta

Columna	transaccion	vendedor	Código	tipo_codigo

Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL
Columna	PK		FK	
Dato de Muestra	1	Juan	1	PERSONA
	2	Pedro	2	EMPRESA
	3	Juan	1	PERSONA

Ejemplo 2.25.

Este ejemplo será realizado utilizando la segunda forma de mapear arcos. El ejemplo se basa en el ejemplo 2.23. El modelo se encuentra en la figura 2.46. El mapeo del supertipo se realizó indicando que cada subtipo seria mapeado como tablas diferentes. Esto ocasiona que la entidad K adquiera un arco de ambos subtipos (Q y R). Esto debido a que ellas heredaron la relación hacia K del supertipo P. El mapeo del arco de la entidad K se realizará utilizando una columna por cada entidad que conforma el arco. El arco se encuentra formado por dos entidades por lo tanto se añadirán dos columnas. Estas columnas serán del tipo opcional.

Tabla 2.24. Mapeo de la entidad K

Columna	id	id_Q	id_R
Opcionalidad	NOT NULL	NULL	NULL
Columna	PK	FK	FK
Dato de Muestra	1		1
	2	1	
	3		1

2.3.1.4. Mapeo de Entidades Recursivas

Las entidades recursivas son entidades que se llaman a si mismas. El mapeo de este tipo de entidades implica añadir una nueva columna que contendrá la llave foránea de esta entidad y que será del tipo opcional. Esto

debido a que de no ser así al momento de realizar una consulta hacia esa tabla puede provocar un ciclo infinito.

Ejemplo 2.26.

El ejemplo se basa en el ejemplo 2.14. y en la figura 2.34. Este ejemplo muestra une entidad recursiva denominada lugar. Esta entidad es recursiva y posee tres atributos. El atributo de nombre, código y tipo de lugar. Al realizar el mapeo se añade además de una columna por estos tres atributos, una columna extra que es la llave foránea recursiva. La tabla 2.25. muestra el mapeo de la entidad recursiva lugar.

Tabla 2.25. Mapeo de la entidad recursiva lugar

Columna	codigo_1	nombre	tipo	codigo_2
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NULL
Columna	PK			FK
Dato de Muestra	1	Área 1	Área	
	2	América	Continente	1
	3	Guatemala	País	2

2.3.1.5. Mapeo de Roles

El mapeo de roles es igual que el mapeo de una entidad normal. El rol se encuentra integrado por una entidad rol y otra entidad relacionada de 1:M que determina el tipo de rol.

Ejemplo 2.26.

El ejemplo se basa en el ejemplo 2.15. el cual contiene un modelo de los diferentes roles que una persona puede desempeñar dependiendo del lugar a la que ésta pertenezca. La figura 2.38. contiene el modelo y los roles definidos para el mismo.

El modelo comprende una entidad rol y una entidad tipo. Esta entidad rol adquiere la llave foránea de la entidad persona, pais y tipo rol. Estas llaves integran la llave primaria de la entidad rol. La tabla 2.26. muestra el mapeo de la entidad rol.

Figura 91. Páginas 53 – 56 Libro “Bases de datos principiantes”

Tabla 2.26. Mapeo de la entidad rol

Columna	codigo_persona	codigo_lugar	codigo_tiporol
Opcionalidad	NOT NULL	NOT NULL	NOT NULL
Columna	PK, FK1	PK, FK2	FK3
Dato de Muestra	1	1	1
	2	2	2
	3	3	3

El mapeo de la entidad tipo rol se muestra en la tabla 2.27.

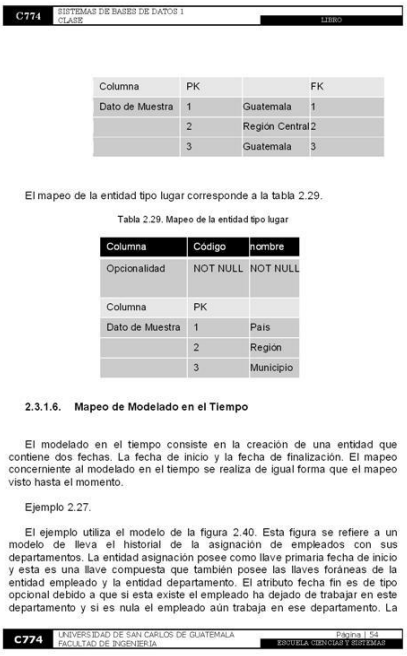
Tabla 2.27. Mapeo de la entidad tipo rol

Columna	codigo	nombre
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	
Dato de Muestra	1	Presidente
	2	Gobernador
	3	Alcalde

La entidad lugar es otro tipo de rol. Esta entidad posee una entidad tipo lugar que indica que tipo de lugar es el dato en la entidad lugar. El mapeo del rol lugar se encuentra dado en la tabla 2.28.

Tabla 2.28. Mapeo del la entidad lugar

Columna	Código	nombre	codigo_tipolugar
Opcionalidad	NOT NULL	NOT NULL	



Columna	PK		FK
Dato de Muestra	1	Guatemala	1
	2	Región Central	2
	3	Guatemala	3

El mapeo de la entidad tipo lugar corresponde a la tabla 2.29.

Tabla 2.29. Mapeo de la entidad tipo lugar

Columna	Código	nombre
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	
Dato de Muestra	1	País
	2	Región
	3	Municipio

2.3.1.6. Mapeo de Modelado en el Tiempo

El modelado en el tiempo consiste en la creación de una entidad que contiene dos fechas. La fecha de inicio y la fecha de finalización. El mapeo concerniente al modelado en el tiempo se realiza de igual forma que el mapeo visto hasta el momento.

Ejemplo 2.27.

El ejemplo utiliza el modelo de la figura 2.40. Esta figura se refiere a un modelo de lleva el historial de la asignación de empleados con sus departamentos. La entidad asignación posee como llave primaria fecha de inicio y esta es una llave compuesta que también posee las llaves foráneas de la entidad empleado y la entidad departamento. El atributo fecha fin es de tipo opcional debido a que si esta existe el empleado ha dejado de trabajar en este departamento y si es nula el empleado aún trabaja en ese departamento. La

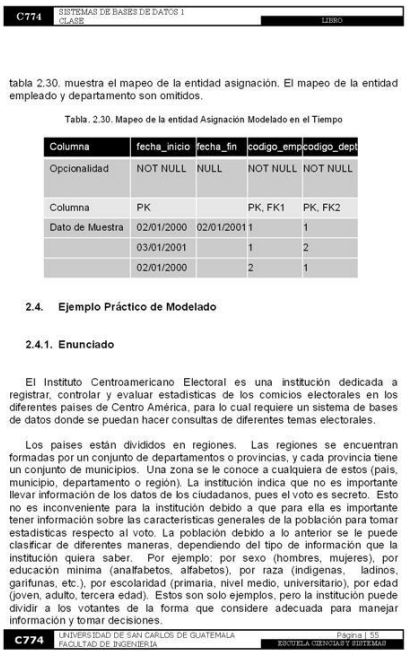


tabla 2.30. muestra el mapeo de la entidad asignación. El mapeo de la entidad empleado y departamento son omitidos.

Tabla. 2.30. Mapeo de la entidad Asignación Modelado en el Tiempo

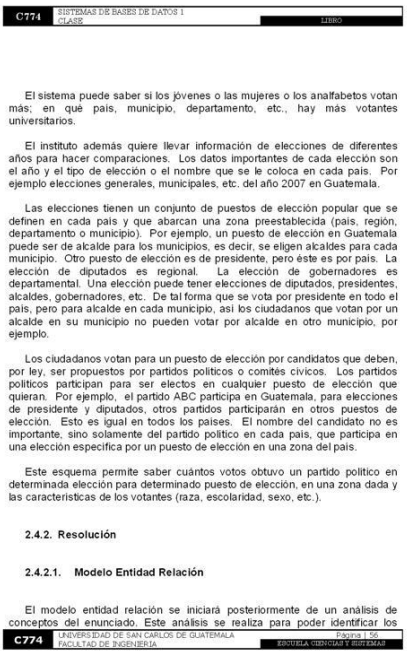
Columna	fecha_inicio	fecha_fin	codigo_emp	codigo_dept
Opcionalidad	NOT NULL	NULL	NOT NULL	NOT NULL
Columna	PK		PK, FK1	PK, FK2
Dato de Muestra	02/01/2000	02/01/2001	1	1
	03/01/2001		1	2
	02/01/2000		2	1

2.4. Ejemplo Práctico de Modelado

2.4.1. Enunciado

El Instituto Centroamericano Electoral es una institución dedicada a registrar, controlar y evaluar estadísticas de los comicios electorales en los diferentes paises de Centro América, para lo cual requiere un sistema de bases de datos donde se puedan hacer consultas de diferentes temas electorales.

Los paises están divididos en regiones. Las regiones se encuentran formadas por un conjunto de departamentos o provincias, y cada provincia tiene un conjunto de municipios. Una zona se le conoce a cualquiera de estos (pais, municipio, departamento o región). La institución indica que no es importante llevar información de los datos de los ciudadanos, pues el voto es secreto. Esto no es inconveniente para la institución debido a que para ella es importante tener información sobre las características generales de la población para tomar estadísticas respecto al voto. La población debido a lo anterior se le puede clasificar de diferentes maneras, dependiendo del tipo de información que la institución quiera saber. Por ejemplo: por sexo (hombres, mujeres), por educación mínima (analfabetos, alfabetos), por raza (indígenas, ladinos, garífunas, etc.), por escolaridad (primaria, nivel medio, universitario), por edad (joven, adulto, tercera edad). Estos son solo ejemplos, pero la institución puede dividir a los votantes de la forma que considere adecuada para manejar información y tomar decisiones.



El sistema puede saber si los jóvenes o las mujeres o los analfabetos votan más; en qué país, municipio, departamento, etc., hay más votantes universitarios.

El instituto además quiere llevar información de elecciones de diferentes años para hacer comparaciones. Los datos importantes de cada elección son el año y el tipo de elección o el nombre que se le coloca en cada país. Por ejemplo elecciones generales, municipales, etc. del año 2007 en Guatemala.

Las elecciones tienen un conjunto de puestos de elección popular que se definen en cada país y que abarcan una zona preestablecida (país, región, departamento o municipio). Por ejemplo, un puesto de elección en Guatemala puede ser de alcalde para los municipios, es decir, se eligen alcaldes para cada municipio. Otro puesto de elección es de presidente, pero éste es por país. La elección de diputados es regional. La elección de gobernadores es departamental. Una elección puede tener elecciones de diputados, presidentes, alcaldes, gobernadores, etc. De tal forma que se vota por presidente en todo el país, pero para alcalde en cada municipio, así los ciudadanos que votan por un alcalde en su municipio no pueden votar por alcalde en otro municipio, por ejemplo.

Los ciudadanos votan para un puesto de elección por candidatos que deben, por ley, ser propuestos por partidos políticos o comités cívicos. Los partidos políticos participan para ser electos en cualquier puesto de elección que quieran. Por ejemplo, el partido ABC participa en Guatemala, para elecciones de presidente y diputados, otros partidos participarán en otros puestos de elección. Esto es igual en todos los países. El nombre del candidato no es importante, sino solamente del partido político en cada país, que participa en una elección específica por un puesto de elección en una zona del país.

Este esquema permite saber cuántos votos obtuvo un partido político en determinada elección para determinado puesto de elección, en una zona dada y las características de los votantes (raza, escolaridad, sexo, etc.).

2.4.2. Resolución

2.4.2.1. Modelo Entidad Relación

El modelo entidad relación se iniciará posteriormente de un análisis de conceptos del enunciado. Este análisis se realiza para poder identificar los

Figura 92. Páginas 57 – 60 Libro “Bases de datos principiantes”

conceptos que se convertirán en entidades. Esto ayuda también a poder situarse en el concepto del enunciado del problema y poder orientar el modelo de tal forma que responda correctamente a las necesidades del negocio.

La regla del negocio más importante corresponde a que no es importante llevar el control sobre los datos personales del votante aunque es importante saber caracteristicas generales del mismo. Esto indica que de alguna forma el modelo debe ser capaz de distinguir poblaciones de votantes conforme a caracteristicas generales.

Los conceptos principales identificados son: población, caracteristica, votación, politico, elección, puesto, pais, región, departamento, municipio y zona.

Los conceptos correspondientes a pais, región, departamento, municipio y zona se refieren a lugares. Estos conceptos pueden ser representados en una entidad recursiva que se denomine lugar. Esto es debido a que pais, región, departamento, municipio y zona no poseen atributos distintivos más que el id, nombre y una descripción.

Las entidades que se obtendrán serán población, politico, elección, puesto y lugar.

La entidad caracteristica ocurre de la interacción de las entidades población con una entidad atributo. Esta entidad atributo permite albergar cualquier atributo que pueda caracterizar una población. Esto origina que se pueda crear una población y a ésta añadir atributos para formar características propias de una población. El modelar así permite que una población x pueda reunir una serie de atributos y éstos formar características que distingan a la población. Esto permite que una población determinada vote en las elecciones. La entidad caracteristica poseerá tres atributos, los cuales serán de tipo carácter, numérico y fecha respectivamente. Esto es debido a que no se sabe con certeza los datos del atributo que se está ingresando y de esta forma se puede soportar los tres tipos de atributos básicos.

Las entidades: politico, puesto, lugar y elección; poseen cada una, una entidad tipo. Esta entidad permitirá llevar de manera fácil el rol de las entidades politico, puesto, lugar y elección. Las entidades tipo de rol poseerán solamente los atributos idtipo y nombtipo. Esto para cada una.

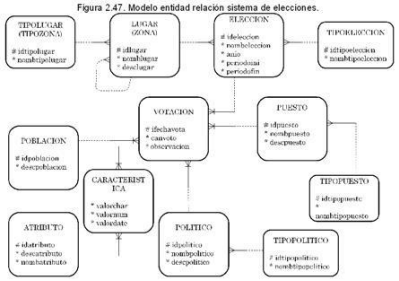
La entidad elección posee su identificador, su atributo nombre, el año en que se realiza la elección, dos atributos que permitirán establecer el periodo inicial de la elección y el periodo final de la elección.

Las entidades lugar, atributo, puesto y politico poseen solamente tres atributos. Estos son el atributo identificador, el atributo nombre y el atributo descripción.

La entidad votación es una entidad que ocurre de la interacción de las entidades población, politico, puesto y elección. Esta entidad posee un atributo que pertenece a la llave primaria de esta entidad. Este atributo es el fecha votación. (fechavota). Este atributo permite indicar la fecha en la que se realizó la votación. Los atributos canvoto permite indicar el número de votos de la población. El atributo observación permite añadir información extra sobre la votación.

Las entidades se encuentran relacionadas de 1:M. Estas donde opcional es del lado de uno y obligatorio del lado de muchos. La figura 2.47. muestra el modelo entidad relación del sistema de elecciones.

Figura 2.47. Modelo entidad relación sistema de elecciones.



2.4.2.2. Mapeo Conceptual

El mapeo conceptual permitirá representar las entidades del modelo entidad relación de elecciones a tablas. Estas tablas contendrán los datos de muestra que permitirán mostrar que tipo de datos pueden almacenarse en cada tabla.

El mapeo se realizará conforme al modelo de la figura 2.47. Este será realizado de derecha a izquierda de arriba hacia abajo.

La entidad TIPOLUGAR se mapeará en la tabla 2.31.

Tabla 2.31. Mapeo entidad TIPOLUGAR

Columna	idtipolugar	nombtipolugar
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	
Dato de Muestra	1	Pais
	2	Departamento
	3	Municipio

La entidad LUGAR se mapeará en la tabla 2.32.

Tabla 2.32. Mapeo entidad LUGAR

Columna	idlugar1	nomblugar	desclugar	idtipolugar	idlugar2
Opcionalidad	NOT NULL	NOT NULL	NULL	NOT NULL	NULL
Columna	PK			FK	
Dato de Muestra	1	Guatemala	Pais	1	
	2	Guatemala	Departamento	2	1
	3	Guatemala	Municipio	3	2

La entidad TIPOELECCION se mapeará en la tabla 2.33.

Tabla 2.33. Mapeo entidad TIPOELECCION

Columna	idtipoeleccion	nombtipoeleccion
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	
Dato de Muestra	1	General
	2	Municipal
	3	Departamental

La entidad ELECCION se mapeará en la tabla 2.34.

Tabla 2.34. Mapeo entidad ELECCION

Columna	ideleccion	nombeleccion	anio	idtipoeleccion	idlugar1	periodoini	periodofin
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NULL	NULL
Columna	PK			FK1	FK2		
Dato de Muestra	1	Elec. Gen. Guatemala	2007	1	1	2008	2012
	2	Muni San Miguel Petapa	2007	2	1	2008	2012
	3	Elec. Gen. Honduras	2006	1	2	2008	2012

Figura 93. Páginas 61 – 64 Libro “Bases de datos principiantes”

La entidad TIPOPUESTO se mapeará en la tabla 2.35.

Tabla 2.35. Mapeo entidad TIPOPUESTO

Columna	idtipopuesto	nombtipopuesto
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	
Dato de Muestra	1	Presidente
	2	Diputado
	3	Alcalde

La entidad PUESTO se mapeará en la tabla 2.36.

Tabla 2.36. Mapeo entidad PUESTO

Columna	idpuesto	nombpuesto	descpuesto	idtipopuesto
Opcionalidad	NOT NULL	NOT NULL	NULL	NOT NULL
Columna	PK			FK
Dato de Muestra	1	Presidente de la República		1
	2	Alcalde Municipal		3
	3	Diputado Departamental		2

La entidad TIPOPOLITICO se mapeará en la tabla 2.37.

Tabla 2.37. Mapeo entidad TIPOPOLITICO

Columna	idtipopolitico	nombtipopolitico
Opcionalidad	NOT NULL	NOT NULL
Columna	PK	
Dato de Muestra	1	Partido Politico

	2	Comité Cívico
	3	Asociación Cívica

La entidad POLITICO se mapeará en la tabla 2.38.

Tabla 2.38. Mapeo entidad POLITICO

Columna	idpolitico	nombpolitico	descpuesto	idtipolitico
Opcionalidad	NOT NULL	NOT NULL	NULL	NOT NULL
Columna	PK			FK
Dato de Muestra	1	PAN		1
	2	GANA		1
	3	PATRIOTA		1

La entidad POBLACION se mapeará en la tabla 2.39.

Tabla 2.39. Mapeo entidad POBLACION

Columna	idpoblacion	descpoblacion
Opcionalidad	NOT NULL	NULL
Columna	PK	
Dato de Muestra	1	Pob. Tipo 1
	2	Pob. Tipo 2
	3	Pob. Tipo 3

La entidad ATRITBUTO se mapeará en la tabla 2.40.

Tabla 2.40. Mapeo entidad ATRIBUTO

Columna	idatributo	nombatributo	tipoatributo

Opcionalidad	NOT NULL	NOT NULL	NOT NULL
Columna	PK		
Dato de Muestra	1	Escolaridad	1 (varchar)
	2	Fecha Nacimiento	3 (date)
	3	Edad	2 (number)

La entidad CARACTERISTICA se mapeará en la tabla 2.41.

Tabla 2.41. Mapeo entidad CARACTERISTICA

Columna	valorchar	valornum	valordate	idpoblacion	idatributo
Opcionalidad	NULL	NULL	NULL	NOT NULL	NOT NULL
Columna	PK			FK1	FK2
Dato de Muestra			18/12/2006	1	2
	Primaria			1	1
	Secundaria			2	1

La entidad VOTACION se mapeará en la tabla 2.42.

Tabla 2.42. Mapeo entidad VOTACION

Columna	ideleccion	idpoblacion	idpolitico	idpuesto	fechavota	observacion	cantvoto
Opcionalidad	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NULL	NOT NULL
Columna	PK, FK1	PK, FK2	PK, FK3	PK, FK4	PK		
Dato de Muestra	1	1	1	1	10/10/2007		1000
	1	2	2	1	10/10/2007		1323

	1	3	1	1	10/10/2007		4321

Figura 94. Páginas 65 – 68 Libro “Bases de datos principiantes”

3. INTRODUCCIÓN A SQL

3.1. Esquema Conceptual

El esquema conceptual es el paso de transformar las tablas obtenidas en el mapeo conceptual a instrucciones SQL. Este paso permite obtener el script de creación de las tablas.

El nombre de las tablas debe ser escrito en plural, debido a que guardan instancias de la entidad. Las llaves primarias se recomienda que sean de tipo numérico.

La instrucción para crear tablas en SQL es “*CREATE TABLE*”. Todas las instrucciones son guardadas en un archivo de texto, que puede ser nombrado con extensión .txt o .sql.

El tipo de los atributos queda sujeto a los soportados por el DBMS que se esté utilizando. Los ejemplos realizados en esta sección se realizarán en el DBMS Oracle.

Ejemplo 2.28.

Se desea crear el esquema conceptual de las entidades de la de la Figura 2.5.

A partir del mapeo conceptual de la Tabla 2.2., 2.3. y 2.4., realizar la construcción del script (esquema conceptual). El script contiene todas las instrucciones correspondientes a SQL. El tipo de los campos dependerá del DBMS utilizado.

La entidad se crea como tabla con las palabras reservadas *CREATE TABLE*. El nombre de la tabla corresponde al nombre de la entidad. Los paréntesis indican que dentro de ellos se deben colocar los campos correspondientes a la tupla de la entidad. El script de la tabla estudiante correspondería a la Figura 3.1. Este se realiza tomando como base el mapeo contenido en la tabla 2.2.

Figura 3.1. Script de creación de la tabla asignación

```
CREATE TABLE estudiante (
```



```
                carnet NUMERIC(7,0) NOT NULL,
                cedula VARCHAR2(9) NULL,
                nombre VARCHAR2(50) NOT NULL,
                telefono NUMERIC(8,0) NOT NULL,
                codigo_curso NUMERIC(4,0) NOT NULL
        );
```

El script anterior contiene solamente la creación de la tabla asignación y de sus campos. Hasta este momento no se han establecido las *constraints*. Ahora se procede a la creación de las restricciones de la entidad estudiante. La creación de estos *constraints* se realizan con las palabras reservadas *ALTER TABLE, ADD CONSTRAINT, PRIMARY KEY, FOREIGN KEY* y *REFERENCES*. La Figura 3.2. presenta el script para añadir los *constraints* a la tabla estudiante.

Figura 3.2. Script de *constraints* de la tabla estudiante

```
ALTER TABLE estudiante ADD CONSTRAINT
        pk_carnet PRIMARY KEY(carnet);
```

Existe otra forma de crear el script dentro de la misma instrucción *CREATE TABLE*. Esta forma permite que los *constraints* sean incorporados después de definir los atributos. La figura 3.3. muestra esta forma de definir el script de la tabla estudiante.

Figura 3.3. Script de creación de la tabla estudiante forma 2

```
CREATE TABLE estudiante (
                carnet NUMERIC(7,0) NOT NULL,
                cedula VARCHAR2(9) NULL,
                nombre VARCHAR2(50) NOT NULL,
                telefono NUMERIC(8,0) NOT NULL,
                codigo_curso NUMERIC(4,0) NOT NULL,
                CONSTRAINT pk_carnet PRIMARY
                        KEY (carnet)
        );
```

El siguiente script corresponderá a la tabla curso. El script de la tabla curso se basa en la tabla 2.3., y correspondería a la Figura 3.4.

Figura 3.4. Script de creación de la tabla curso

```
CREATE TABLE curso (
                codigo NUMERIC(7,0) NOT NULL,
                nombre VARCHAR2(50) NOT NULL,
                descripcion VARCHAR2(150) NOT NULL,
        );
```

La Figura 3.5. presenta el script para añadir los *constraints* a la tabla curso.

Figura 3.5. Script de *constraints* de la tabla curso

```
ALTER TABLE curso ADD CONSTRAINT
        pk_cursoCod PRIMARY KEY(codigo);
```

El siguiente script corresponderá a la tabla asignación. El script de la tabla asignación se basa en la tabla 2.4., y correspondería a la Figura 3.6.

Figura 3.6. Script de creación de la tabla asignación

```
CREATE TABLE asignacion (
                no_asignacion NUMERIC(7,0) NOT NULL,
                estado VARCHAR2(1) NOT NULL,
                semestre VARCHAR2(5) NOT NULL,
                carnet_estudiante NUMERIC(8,0) NOT NULL,
                codigo_curso NUMERIC(4,0) NOT NULL
        );
```

La Figura 3.7. presenta el script para añadir los *constraints* a la tabla asignación.

Figura 3.7. Script de *constraints* de la tabla asignación

```
ALTER TABLE asignacion ADD CONSTRAINT
        pk_asignacion PRIMARY KEY(no_asignacion);
ALTER TABLE asignacion ADD CONSTRAINT
        fk1_asignacion FOREIGN KEY (carnet_estudiante)
                REFERENCES estudiante (carnet);
ALTER TABLE asignacion ADD CONSTRAINT
        fk2_asignacion FOREIGN KEY (codigo_curso)
                REFERENCES curso (codigo);
```

Ejemplo 2.29.

El ejemplo se basa en la figura 2.19. que corresponde a arrastrar la llave primaria. Este modelo se encuentra integrado por tres entidades. El mapeo de estas entidades se encuentra en las tablas 2.8., 2.9. y 2.10.

El script que corresponde a la tabla empleado (tabla 2.8.), se representa en la figura 3.8. Esta tabla no posee ninguna llave foránea.

Figura 3.8. Script de creación de la tabla empleado

```
CREATE TABLE empleado (
                carnet NUMERIC(7,0) NOT NULL,
                cedula VARCHAR2(9) NULL,
                nombre VARCHAR2(50) NOT NULL,
                puesto VARCHAR2(25) NOT NULL,
                fecha_nac DATE NOT NULL,
                CONSTRAINT pk_empleado PRIMARY
```



```
                        KEY (carnet)
        );
```

El script que corresponde a la tabla factura (tabla 2.9.), se representa en la figura 3.9. Esta tabla posee una llave foránea de empleado que forma parte de la llave primaria de factura.

Figura 3.9. Script de creación de la tabla factura

```
CREATE TABLE factura (
                carnet_emp NUMERIC(7,0) NOT NULL,
                numero NUMERIC(7,0) NOT NULL,
                fecha DATE NOT NULL,
                cliente VARCHAR2(50) NOT NULL,
                CONSTRAINT pk_empleado PRIMARY
                        KEY (carnet, numero),
                CONSTRAINT fk_emp leado FOREIGN
        KEY
                        (carnet_emp) REFERENCES
                                empleado(carnet)
        );
```

El script que corresponde a la tabla detalle_factura (tabla 2.10.), se representa en la figura 3.10. Esta tabla posee una llave foránea de empleado que forma parte de la llave primaria de factura.

Figura 3.10. Script de creación de la tabla detalle_factura

```
CREATE TABLE detalle_factura (
                carnet_emp NUMERIC(7,0) NOT NULL,
                numero NUMERIC(7,0) NOT NULL,
                linea NUMERIC(7,0) NOT NULL,
                producto VARCHAR2(50) NOT NULL,
                cantidad NUMERIC(7,0) NOT NULL,
                CONSTRAINT pk_detalle_factura PRIMARY
                        KEY (carnet, numero, linea),
                CONSTRAINT fk_detalle_factura FOREIGN
        KEY
                        (carnet_emp, numero)
        REFERENCES
                                factura(carnet_emp,
                        numero)
        );
```

Ejemplo 2.30.

El ejemplo se basa en la figura 2.34. que corresponde a una entidad recursiva. Este modelo se encuentra integrado por la entidad lugar, la cual es recursiva. El mapeo de esta entidad se encuentra en las tablas 2.25.

Figura 95. Páginas 69 – 72 Libro “Bases de datos principiantes”

El script que corresponde a la tabla empleado (tabla 2.35.), se representa en la figura 3.11. Esta tabla no posee ninguna llave foránea.

Figura 3.11. Script de creación de la tabla lugar (recursiva)

```
CREATE TABLE lugar (
                codigo_1 NUMERIC(7,0) NOT NULL,
                nombre VARCHAR2(50) NULL,
                tipo VARCHAR2(20) NOT NULL,
                codigo_2 NUMERIC(7,0) NOT NULL,
                CONSTRAINT pk_lugar PRIMARY
                        KEY (codigo_1),
                CONSTRAINT fk_lugar FOREIGN
        KEY
                        (codigo_2) REFERENCES
                                lugar(codigo_1),
        );
```

3.2. SQL Básicos de Creación de Tablas

Los comandos básicos de creación de tablas son cinco. Estos comandos son:

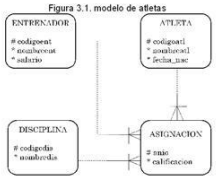
- *CREATE TABLE*: este comando se utilizó en la sección 3.1. Este comando permite crear una tabla.
- *DROP TABLE*: este comando permite eliminar una tabla. El comando elimina cualquier registro que se encuentre dentro de la tabla que se está eliminando. La sintaxis es DROP TABLE NombreTabla.
- *CONSTRAINT*: este comando se utiliza para indicar restricciones sobre los atributos de las tablas. Esto quiere decir que se utiliza para añadir una restricción de llave primaria, foránea ó única. La restricción es evaluada y si ocurre una violación esta se despliega con el nombre que se le haya dado al *CONSTRAINT*.
- *PRIMARY KEY*: este comando se utiliza para indicar que el atributo será utilizado como una llave primaria.
- *FOREIGN KEY*: este comando se utiliza para indicar que el atributo será utilizado como una llave foránea.

Ejemplo 3.1.

El ejemplo 3.1. será utilizado como base para todo lo que se realice en la sección 3.

El modelo que se utilizará será uno concerniente a un sistema que permite llevar el control de los atletas, las asignaciones que éstos realizan con respecto a disciplinas y la persona que los ha entrenado en esa disciplina. La figura 3.1. muestra el modelo de atletas.

Figura 3.1. modelo de atletas



La creación de las tablas quedará plasmada en la figura 3.2. Esta contiene los comandos para la creación de todas las tablas del modelo de atletas.

Figura 3.2. Script de creación de la tabla factura

```
CREATE TABLE entrenador (
                codigoent NUMERIC(7,0) NOT NULL,
                nombreent VARCHAR2(50) NOT NULL,
                salario NUMERIC( 10,2) NOT NULL,
                CONSTRAINT pk_ent PRIMARY
                        KEY (codigoent)
        );

CREATE TABLE atleta (
                codigoatl NUMERIC(7,0) NOT NULL,
                nombreatl VARCHAR2(50) NOT NULL,
                fecha_nacatl DATE NOT NULL,
                CONSTRAINT pk_atl PRIMARY
                        KEY (codigoatl)
        );

CREATE TABLE disciplina (
                codigodis NUMERIC(7,0) NOT NULL,
                nombredis VARCHAR2(50) NOT NULL,
                CONSTRAINT pk_dis PRIMARY
                        KEY (codigodis)
        );

CREATE TABLE asignacion (
                calificacion NUMERIC(7,0) NOT NULL,
                anio NUMERIC(4,0) NOT NULL,
```



```
                codigoatl NUMERIC(7,0) NOT NULL,
                codigoent NUMERIC(7,0) NOT NULL,
                codigodis NUMERIC(7,0) NOT NULL,
                CONSTRAINT fk_asient FOREIGN
        KEY
                        (codigoent) REFERENCES
                                entrenador(codigoe
                                nt),
                CONSTRAINT fk_asiatl FOREIGN KEY
                        (codigoatl) REFERENCES
                                atleta(codigoatl),
                CONSTRAINT fk_asidis FOREIGN KEY
                        (codigodis) REFERENCES
                                disciplina(codigodis
                                ),
                CONSTRAINT pk_asig PRIMARY
                        KEY (codigoatl, codigoent,
                codigodis),
        );
```

3.3. Consultas

Las consultas son métodos que permiten acceder a los datos de las bases de datos. Esto quiere decir que se consulta la información que se encuentra dentro de las tablas de los modelos de datos. Las consultas permiten la modificación, eliminación, visualización y agregación de los datos en la base de datos. Las consultas se realizan por el lenguaje de consultas SQL (*structured query language*).

El lenguaje SQL se basa en sentencias. Las sentencias SQL se pueden dividir en dos grupos principales. Estos son:

- DDL: estas son las siglas para lenguaje de definición de datos. Las sentencias DDL crean objetos en la base de datos. Los efectos de esta creación se reflejan en el diccionario de datos. Las sentencias DDL básicas son: *CREATE TABLE, DROP TABLE, ALTER TABLE, GRANT, TRUNCATE, REVOKE*.
- DML: estas son las siglas para lenguaje de manipulación de datos. Las sentencias DML permiten consultar, insertar, modificar y eliminar la información que se encuentra dentro de los objetos de la base de datos. Las sentencias DML básicas son:
  - *INSERT*: esta sentencia permite añadir a una tabla una fila de datos.
  - *DELETE*: esta sentencia permite eliminar en una tabla una fila o filas de datos.
  - *UPDATE*: esta sentencia permite modificar en una tabla uno fila o filas de datos.

  - *SELECT*: esta sentencia permite realizar una consulta de datos de una o más tablas.
  - *COMMIT*: esta sentencia permite confirmar las modificaciones permanentemente.
  - *ROLLBACK*: esta sentencia permite deshacer las modificaciones realizadas a partir de la última confirmación.

3.3.1. Proyecciones

Una proyección ocurre cuando se restringe el número de campos mostrados en una o más tablas. Esto permite indicar los campos que se quieren mostrar.

Las consultas empiezan básicamente con dos cláusulas:

- *SELECT*: esta cláusula se utiliza para especificar los campos que se necesitan de una tabla. Los campos se definen posteriormente de la cláusula, si se coloca en lugar del nombre de los campos el carácter *, se listarán todos los campos de todas las tablas que se especifican en *FROM*.
- *FROM*: esta cláusula se utiliza para especificar las tablas de donde se están obteniendo los datos.

Ejemplo 3.2.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar la proyección de la tabla atleta presentando el código y el nombre de los atletas. La figura 3.3. muestra el script de la proyección de esta tabla. Esta listará todos los atletas que se encuentren en la tabla atleta, mostrando únicamente el código y el nombre para cada atleta.

Figura 3.3. Script de proyección de la tabla atleta

```
SELECT codigoatl, nombreatl
FROM atleta
;
```

Figura 96. Páginas 73 – 76 Libro “Bases de datos principiantes”

CODIGOATL	NOMBREATL
1	A
2	B
3	C
4	D
5	E
6	F
7	G

3.3.2. Restricciones

Una restricción se realiza con el fin de limitar el número de registros que se obtienen al hacer una consulta. Esto quiere decir que las filas a mostrar son limitadas por medio de una restricción.

Las consultas que restringen los datos utilizan la clausula *WHERE*. Las restricciones pueden ser divididas por medio de conectores lógicos (*AND* y *OR*). Las restricciones en el *WHERE* deben devolver valores de verdadero y falso.

Ejemplo 3.3.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., mostrar los datos del atleta con código 3. Esta consulta permite mostrar el nombre y la fecha de nacimiento para el atleta con código 3. Esto permite restringir la consulta para que muestre únicamente los datos del atleta 3 e ignore a los demás atletas. La figura 3.4. muestra la consulta que restringe al atleta 3.

Figura 3.4. Script de restricción del atleta 3

```
SELECT nombreatl, fecha_nacatl
FROM atleta
WHERE codigoatl = 3
;
```

HOMBREATL	FECHA_NACATL
C	12/12/78

3.3.2.1. Operadores Matemáticos

Los operadores matemáticos son utilizados en otros lenguajes de programación comunes. Estos permiten la creación de restricciones necesarias para obtener los resultados deseados en las consultas. Los operadores matemáticos son:

- Igual: =.
- Menor ó mayor: <, >.
- Menor igual ó mayor igual: <=, >=.
- Diferente a: <>.

Ejemplo 3.4.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre los entrenadores cuyo salario es menor a 3000. La figura 3.5. muestra la consulta que muestra el listado de los entrenadores que ganan menos de 3000.

Figura 3.5. Script de proyección de la tabla atleta

```
SELECT e.codigoent entrenador, e.nombreent nombre, e.salario salario
FROM entrenador e
WHERE e.salario < 3000;
```

ENTRENADOR	NOMBRE	SALARIO
1	Juan Rosito Guttelewitz	1500
2	James Lopez	2505
3	Gabino Gainza	1200,55
4	Fernando Ruiz	2500
5	Carlos Gomez	1800

3.3.2.2. Operadores Lógicos

Los operadores lógicos son utilizados en las consultas para unir dos o más condiciones que integran la restricción. Los operadores utilizados son el AND y el OR. Estos se mezclan con los operadores matemáticos para formar restricciones más complejas.

Ejemplo 3.5.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre los entrenadores que poseen un salario que se encuentra entre 1000 y 2000; además que muestre los entrenadores que ganan exactamente 3000. La figura 3.6. muestra esta consulta.

Figura 3.6. Script de consulta de rango de salarios

```
SELECT e.codigoent entrenador, e.nombreent nombre, e.salario salario
FROM entrenador e
WHERE e.salario > 1000
AND e.salario < 2000
OR e.salario = 3000;
```

ENTRENADOR	NOMBRE	SALARIO
1	Juan Rosito Guttelewitz	1500
3	Gabino Gainza	1200,55
5	Carlos Gomez	1800
6	Luisa Martinez	3000
7	Mario Juarez	3000

3.3.3. *Aliasing*

Aliasing es una forma de identificar de manera única las tablas que se seleccionan en la cláusula FROM. Esto permite no tener que utilizar el nombre completo de la tabla ó para hacer diferencia entre tablas que procedan de la misma tabla. El *aliasing* también es utilizado para cambiar el nombre de los campos que se seleccionan. Esto último es útil cuando se están utilizando funciones y se le da un nombre a esta función con el *aliasing*. La diferencia entre el *aliasing* de tablas y de campos es que cuando se utiliza en éstos últimos se utiliza la palabra reservada AS. Los campos de una tabla en la cual se ha utilizado *aliasing* se referencian por el nombre que se le ha dado a la tabla dentro de la cláusula *FROM* seguido de punto y el nombre del atributo.

Ejemplo 3.6.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar mostrar el código y el nombre de los atletas. El código debe llamarse atleta y el nombre será nombre_atleta. La tabla atleta se renombrará como atl.

La figura 3.7. muestra la consulta que muestra el listado de atletas utilizando el *aliasing*.

Figura 3.7. Script de proyección de la tabla atleta

```
SELECT atl.codigoatl as atleta, atl.nombreatl as nombre_atleta
FROM atleta atl
;
```

ATLETA	NOMBRE_ATLETA
1	A
2	B
3	C
4	D
5	E
6	F
7	G

3.3.4. Insertar y Eliminar Datos

La inserción de datos se realiza por medio de la sentencia INSERT. La sintaxis es: *INSERT into* {tabla} values ({val1},{val2},…,{valn});. Esta forma inserta los datos en el orden en que los campos fueron creados en la tabla, si un dato no se coloca en ese orden desplegará una violación en el caso de que no sean del mismo tipo. Otra forma para insertar pero que permite indicar el orden de los campos es la siguiente: *INSERT into* {tabla} ({campo 1},{campo 2},…,{campo n}) values ({val1},{val2},…,{valn});. Esta forma puede hasta omitir algún campo en el caso de que pueda ser nulo.

La instrucción *DELETE* permite eliminar campos de una tabla. La instrucción necesita de una condición. Esto quiere decir que utiliza una restricción para poder realizar la eliminación de los registros de una tabla. La sintaxis es: *DELETE from* {tabla} *WHERE* {condición}. La condición es una condición de una consulta normal.

La instrucción *UPDATE* permite modificar campos de una tabla. La instrucción al igual que la eliminación necesita de una condición. La sintaxis es: *UPDATE* {tabla} *SET* {campo(s) = valor} *WHERE* {condición}.

Figura 97. Páginas 77 – 80 Libro “Bases de datos principiantes”

Ejemplo 3.7.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar la inserción correspondiente a cada una de las tablas creadas. Al terminar la inserción elimine las asignaciones del atleta 1 y luego al atleta 1. Al finalizar la eliminación, modifique el salario del entrenador 1 y colóquele un salario de 2000. Los datos insertados son solamente de ejemplo para el uso de la instrucción *INSERT*. La figura 3.8. muestra los *insert, delete* y *update*.

Figura 3.8. Script de proyección de la tabla atleta

```
INSERT into ATLETA (codigoatl,nombreatl,fecha_nacatl)
 values (1,'A','05/03/1970');

INSERT into ENTRENADOR (codigoent,nombreent,salario)
 values (1,'Juan Rosito Guttelewitz',1500.00);

INSERT into DISCIPLINA (codigodis,nombredis)
 values (1,'Tiro');

INSERT into ASIGNACION (codigoent,codigoatl,codigodis,anio,calificacion)
 values (1,1,1,2005,73);

DELETE FROM atleta
WHERE codigoatl = 1;

UPDATE entrenador
SET salario = 2000
WHERE codigoent = 1;
```

3.3.5. Funciones

Los DBMS incluyen una serie de funciones que facilitan realizar consultas. Generalmente muchos de ellos incluyen una serie de funciones básicas que son comúnmente utilizadas. Las cinco funciones básicas son:

- *SUM*: esta función suma los valores numéricos contenidos en el campo seleccionado.
- *COUNT*: esta función suma el número de registros contenido en la consulta realizada.
- *AVG*: esta función realiza un promedio de los valores numéricos del campo seleccionado.
- *MIN*: esta función encuentra el registro con el valor más bajo en el campo seleccionado.
- *MAX*: esta función encuentra el registro con el valor más alto en el campo seleccionado.

Las funciones se utilizan en la parte del *SELECT*. El atributo a mostrar en el *SELECT*, si es solo la función se realiza como una consulta normal. El problema radica cuando se añade otro campo además de la función. Esto quiere decir que el campo o campos que se añaden serán en conjunto los que determinen el valor de la función. La función se ejecutará correspondiente al conjunto de campos. Esto significa que se tendrá que añadir una cláusula denominada *GROUP BY*. Esta clausula permitirá agrupar el conjunto de datos y la función se ejecutará a partir de estos. El GROUP BY utiliza el nombre de los campos no el alias.

Ejemplo 3.8.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que cuente la cantidad de entrenadores que poseen un salario menor a 3000 . La figura 3.9. muestra esta consulta.

Figura 3.9. Script de consulta de ejemplo de *count*

```
SELECT COUNT (e.codigoent) AS cantidad_entrenadores
FROM entrenador e
WHERE e.salario < 3000
.
```

CANTIDAD_ENTRENADORES
5

Ejemplo 3.9.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que sume los salarios de todos los entrenadores que ganan menos de 3000. La figura 3.10. muestra esta consulta.

Figura 3.10. Script consulta de ejemplo de *sum*

```
SELECT SUM (e.salario) AS total_salario_entrenadores
FROM entrenador e
WHERE e.salario < 3000
.
```

TOTAL_SALARIO_ENTRENADORES
9505,55

Ejemplo 3.10.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta donde se muestre el promedio de los salarios de los

entrenadores que ganan menos de 3000. La figura 3.11. muestra el script de esta consulta.

Figura 3.11. Script consulta de ejemplo de *avg*

```
SELECT AVG (e.salario) AS prom_salario_entrenadores
FROM entrenador e
WHERE e.salario < 3000
.
```

PROM_SALARIO_ENTRENADORES
1901,11

Ejemplo 3.11.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre el mayor salario, de los salarios menores a 3000. La figura 3.12. muestra la consulta que despliega el mayor salario.

Figura 3.12. Script consulta ejemplo *max*

```
SELECT MAX (e.salario) AS maximo_salario
FROM entrenador e
WHERE e.salario < 3000
.
```

MAXIMO_SALARIO
2505

Ejemplo 3.12.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre el menor salario, de los salarios menores a 3000. La figura 3.12. muestra la consulta que despliega el mayor salario.

Figura 3.13. Script

```
SELECT MIN (e.salario) AS minimo_salario
FROM entrenador e
WHERE e.salario < 3000
.
```

MINIMO_SALARIO
1200,55

Ejemplo 3.13.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que despliegue la cantidad de veces que un atleta ha sido entrenado por un entrenador, despliegue solamente los códigos de cada uno. La consulta se hace haciendo referencia a la tabla asignación y se utilizará la función COUNT agrupada con el GROUP BY. La figura 3.14. muestra la consulta.

Figura 3.14. Script ejemplo consulta *group by*

```
SELECT asi.codigoatl as atleta, asi.codigoent as entrenador, COUNT (asi.codigoatl) as no_veces
FROM asignacion asi
GROUP BY asi.codigoatl, asi.codigoent
.
```

ATLETA	ENTRENADOR	NO_VECES
1	1	2
1	2	2
1	3	1
2	1	1
2	2	1
2	3	1
3	1	1
3	3	2
3	6	1
4	2	1
4	6	1
5	1	1
5	2	1
5	4	1
6	4	1
6	5	1
7	5	2

3.3.6. *JOINS*

Los *JOINS* permiten en una consulta unir los datos de una o más tablas. Las tablas que se unen son las que poseen campos foráneos. Esto quiere decir en el caso de los atletas, que se puede realizar un *JOIN* entre la tabla atleta y la

Figura 98. Páginas 81 – 84 Libro “Bases de datos principiantes”

tabla asignación, porque la tabla asignación posee como atributo foráneo el código del atleta.

El problema del uso inadecuado de los *JOINS* es que se puede tener el producto cartesiano de las dos tablas que se están uniendo. Esto sería un problema debido a que habrían datos inconsistentes ya que cada campo de la tabla dos se asociaría con cada campo de la tabla dos.

Los DBMS presentan distintas disposiciones ya sea como OUTER JOIN, INNER JOIN, pero en esta sección se utilizarán JOINS utilizando restricciones que igualen ambos campos de las tablas y de esta forma poder crear otra que contenga los datos de las mismas. Las tablas que se pueden unir en una consulta no tienen límite más que estén relacionadas por llaves foráneas. Esto es para mantener la integridad de los datos que se presenten.

Ejemplo 3.14.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre las asignaciones de los atletas mostrando su nombre, junto con el nombre de la persona que los entrenó, el nombre de la disciplina y la calificación de los atletas. La consulta se realizará utilizando *JOINS*, entre las tabla atleta, entrenador y disciplina con la tabla asignación. Esto debido a que la tabla asignación en sus campos posee referencia foránea a las tablas atleta, entrenador y disciplina. La figura 3.15. muestra la forma de realizar un *JOIN*.

Figura 3.15. Script ejemplo de uso de *JOIN*

```
SELECT ati.nombreati AS atleta, ent.nombreent AS entrenador,
dis.nombredis AS disciplina, asi.calificacion, asi.ano AS año
FROM atleta ati, entrenador ent, disciplina dis, asignacion asi
WHERE ati.codigoati = asi.codigoati
AND ent.codigoent = asi.codigoent
AND dis.codigodis = asi.codigodis
;
```


ATLETA	ENTRENADOR	DISCIPLINA	CALIFICACION	AÑO
A	Juan Rosito Guttelewitz	Tiro	73	2005
A	Juan Rosito Guttelewitz	Carrera	85	2005
A	James Lopez	Ecuestre	90	2005
A	James Lopez	Esgrima	95	2005
A	Gabino Gainza	Natacion	92	2005
B	Juan Rosito Guttelewitz	Tiro	60	2004
C	Juan Rosito Guttelewitz	Carrera	85	2005
D	James Lopez	Ecuestre	90	2005
B	James Lopez	Tiro	95	2005
C	Gabino Gainza	Natacion	10	2005
E	Fernando Ruiz	Tiro	73	2005
F	Fernando Ruiz	Tiro	85	2005
F	Carlos Gomez	Carrera	90	2005
G	Carlos Gomez	Esgrima	65	2005
G	Carlos Gomez	Natacion	90	2005
E	Juan Rosito Guttelewitz	Natacion	60	2004
E	James Lopez	Carrera	85	2005
D	Luisa Martinez	Tiro	90	2005
C	Luisa Martinez	Carrera	85	2005
B	Gabino Gainza	Carrera	37	2005
C	Gabino Gainza	Carrera	90	2005

Ejemplo 3.15.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre la mayor nota que se ha obtenido para cada disciplina. La consulta se realizará con un JOIN entre la tabla disciplina y asignación. La figura 3.16. muestra esta consulta.

Figura 3.16. Script ejemplo de *join* utilizando funciones

```
SELECT dis.nombredis AS disciplina, MAX(asi.calificacion) AS max_calificacion
FROM disciplina dis, asignacion asi
WHERE dis.codigodis = asi.codigodis
GROUP BY dis.nombredis
;
```


DISCIPLINA	MAX_CALIFICACION
Carrera	90
Natacion	92
Tiro	95
Esgrima	95
Ecuestre	90

3.3.7. Campos Calculados

Los campos calculados son campos que se seleccionan en una consulta pero que están acompañados de valores estáticos.

Los campos que se pueden calcular son campos devueltos o funciones. Esto quiere decir que una función se le puede añadir algún operador estático que permita realizar un cálculo. El restarle algún valor de penalización a una calificación ó sumarle un bono a alguna nota por ser la de mayor punteo, son algunos ejemplos del uso de campos calculados.

Otra forma de utilizar campos es añadir un campo estático que brinde información extra a los campos que se devuelven en una consulta. Este campo se coloca entre apóstrofes. Esto permite colocar cualquier palabra o frase y añadirla a la consulta.

Ejemplo 3.16.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva el IGSS que los entrenadores deben pagar con respecto a su salario. El pago de IGSS es el 3.5% del salario de una persona. El salario se conoce entonces bastará con devolver cada entrenador y sacar el 3.5% a su salario. La figura 3.17. muestra esta consulta.

Figura 3.17. Script consulta ejemplo de campo calculado estático

```
SELECT ent.codigoent as entrenador, ent.nombreent AS nombre,
ent.salario*0.035 AS IGSS
FROM entrenador ent
;
```


ENTRENADOR	NOMBRE	IGSS
1	Juan Rosito Guttelewitz	52,5
2	James Lopez	87,675
3	Gabino Gainza	42,01925
4	Fernando Ruiz	87,5
5	Carlos Gomez	63
6	Luisa Martinez	105
7	Mario Juarez	105

Ejemplo 3.17.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre la mayor calificación obtenida por todos los atletas y se le debe restar 15 puntos de penalización. La consulta se realizará uniendo las tablas atleta y asignación y posteriormente se utilizará la función *MAX*. La figura 3.18. muestra esta consulta.

Figura 3.18. Script consulta ejemplo uso de función con campo calculado

```
SELECT ati.codigoati AS atleta, ati.nombreati AS nombre_atleta,
MAX(asi.calificacion)-15 AS calificación_max
FROM atleta ati, asignacion asi
WHERE ati.codigoati = asi.codigoati
GROUP BY ati.codigoati, ati.nombreati
;
```

ATLETA	HOMBRE_ATLETA	CALIFICACIÓN_MAX
1	A	80
4	D	75
6	F	75
5	E	70
7	G	75
2	B	80
3	C	75

Ejemplo 3.18.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que liste todos los entrenadores y que les conceda el título de entrenador titular. La figura 3.19. muestra esta consulta.

Figura 3.19. Script consulta ejemplo campo estático

```
SELECT ent.codigoent as entrenador, ent.nombreent as nombre,
'entrenador titular'
FROM entrenador ent
```

Figura 99. Páginas 85 – 88 Libro “Bases de datos principiantes”


```
.
```

ENTRENADOR	NOMBRE	'ENTRENADORTITULAR'
1	Juan Rosito Guttelewitz	entrenador titular
2	James Lopez	entrenador titular
3	Gabino Gainza	entrenador titular
4	Fernando Ruiz	entrenador titular
5	Carlos Gomez	entrenador titular
6	Luisa Martinez	entrenador titular
7	Mario Juarez	entrenador titular

3.3.8. Subconsultas

Una subconsulta es generalmente una consulta que se encuentra contenida en otra consulta. Técnicamente una subconsulta es una instrucción select anidada dentro de un *select*, *update*, *insert* o *delete* que se coloca dentro de paréntesis. Existen diferentes formas de utilizar subconsulta. Las formas más comunes es utilizarlas dentro de las cláusulas *FROM* o *WHERE*.

Las subconsultas exigen que se utilice el *aliasing*. Esto es debido a que las subconsultas se encuentran dentro de otras consultas. Esto ocasiona que muchas se haga referencia a las mismas tablas y esto ocasiona que los campos posean el mismo nombre de la misma tabla y el DBMS no sea capaz de entender a cual campo se refiere y de que tabla.

3.3.9. Subconsultas Usando *WHERE*

Las subconsultas usando *where* son subconsultas que se realizan dentro de la cláusula *WHERE*. Estas subconsultas se ejecutan cada vez que se realiza la restricción en la cual se añade la subconsulta. Los campos que se devuelven en la subconsulta deben corresponder al tipo de condición realiza. Esto quiere decir que si se está buscando un valor mayor a un campo la subconsulta debe devolver un valor numérico. Otra forma es que si el valor del campo a comparar se encuentra en un conjunto de datos se debe devolver más de un dato la subconsulta.

Ejemplo 3.19.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre a los entrenadores con un salario igual al entrenador con código 6. Esta consulta se debe realizar utilizando una subconsulta. La subconsulta permitirá encontrar el salario del entrenador con código 6. Una vez encontrado este valor se procederá a comparar cada uno de los salarios de todos los entrenadores en la consulta principal. La consulta principal posee además una restricción que indica que se debe obviar el entrenador 6 debido a que es él con el que se está comparando. La figura 3.20. muestra esta consulta.

Figura 3.20. Script ejemplo subconsulta *where* 1

```
SELECT ent.codigoent AS entrenador, ent.nombreent AS nombre
FROM entrenador ent
WHERE ent.codigoent <> 6
AND   ent.salario = (
                SELECT ent2.salario
                FROM entrenador ent2
                WHERE ent2.codigoent = 6
                )
.
```

ENTRENADOR	NOMBRE
7	Mario Juarez

Ejemplo 3.20.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva la disciplina, el atleta y la calificación. Esto para el atleta que ha obtenido la mejor calificación en esa disciplina. La consulta se divide en dos partes. La primer parte necesita de una subconsulta que muestre la nota máxima de una disciplina. La consulta principal se encargará de brindar el *join* de las tablas atleta, disciplina y calificación y posteriormente comparar la máxima nota que devuelva la subconsulta. Esto trabaja bien el único inconveniente es que la subconsulta así estructurada dará la mayor nota de todas las disciplinas. Esto se evita utilizando el código de la disciplina de la consulta principal y utilizarla en la restricción de la subconsulta para que el valor máximo que se compara es el de la disciplina actual. La figura 3.21. muestra esta consulta.

Figura 3.21. Script ejemplo subconsulta *where* 2

```
SELECT atl.codigoatl AS atleta, atl.nombreatl AS nombre_atleta,
dis.nombredis AS nombre_disciplina, asi.calificacion AS nota_maxima
FROM atleta atl, disciplina dis, asignacion asi
WHERE atl.codigoatl = asi.codigoatl
```



```
AND dis.codigodis = asi.codigodis
AND asi.calificacion = (
                SELECT MAX(asi2.calificacion)
                FROM asignacion asi2
                WHERE dis.codigodis = asi2.codigodis
                )
.
```

ATLETA	NOMBRE_ATLETA	NOMBRE_DISCIPLINA	NOTA_MAXIMA
1	A	Ecuestre	90
1	A	Esgrima	95
1	A	Natacion	92
4	D	Ecuestre	90
2	B	Tiro	95
6	F	Carrera	90
3	C	Carrera	90

3.3.10. Subconsultas Usando *FROM*

Una subconsulta usando *FROM* es una subconsulta que se realiza en la cláusula *FROM*. Este tipo de subconsulta difiere de la subconsulta de *WHERE* con que la subconsulta se ejecutará una vez y se le añadirá un nombre mientras la ejecución de la consulta principal dure. Este tipo de subconsultas debe añadírsele un alias luego de realizada. Los campos de la subconsulta se recomienda que se les añadan alias también para evitar problemas de entendimiento tanto del DBMS y del programador.

Ejemplo 3.21.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta utilizando subconsultas de FROM, que devuelva el listado de entrenadores con salario igual al entrenador con código 6. Esta consulta se asemeja a la del ejemplo 3.19. la diferencia es que la subconsulta debe devolver el código y el salario del entrenador con código 6. Esta consulta se utilizará como si fuera una tabla más y tendremos que utilizarla junto con las otras tablas de la consulta principal. La figura 3.22. muestra esta consulta.

Figura 3.22. Script ejemplo subconsulta de *from* 1

```
SELECT ent.codigoent AS entrenador, ent.nombreent AS nombre_entrenador,
ent.salario AS salario_entrenador
FROM entrenador ent, (
                SELECT ent2.codigoent as codigoent, ent2.salario AS salario
                FROM entrenador ent2
```



```
                WHERE ent2.codigoent = 6
) tabent
WHERE ent.codigoent <> 6
AND ent.salario = tabent.salario
.
```

ENTRENADOR	NOMBRE_ENTRENADOR	SALARIO_ENTRENADOR
7	Mario Juarez	3000

Ejemplo 3.22.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva la disciplina, el atleta y la calificación. Esto para el atleta que ha obtenido la mejor calificación en esa disciplina. La consulta se divide en dos partes. La figura 3.23..

Figura 3.23. Script

```
SELECT atl.codigoatl AS atleta, atl.nombreatl AS nombre_atleta,
dis.nombredis AS nombre_disciplina, asi.calificacion AS nota_maxima
FROM atleta atl, disciplina dis, asignacion asi,
(
        SELECT asi2.codigodis as codigodis, MAX(asi2.calificacion) AS calif_max
        FROM asignacion asi2
        GROUP BY asi2.codigodis
) tabmax
WHERE atl.codigoatl = asi.codigoatl
AND dis.codigodis = asi.codigodis
AND tabmax.codigodis = dis.codigodis
AND tabmax.calif_max = asi.calificacion
.
```

ATLETA	NOMBRE_ATLETA	NOMBRE_DISCIPLINA	NOTA_MAXIMA
1	A	Ecuestre	90
1	A	Esgrima	95
1	A	Natacion	92
4	D	Ecuestre	90
2	B	Tiro	95
6	F	Carrera	90
3	C	Carrera	90

Ejemplo 3.23.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva el nombre del entrenador, el nombre del atleta y la calificación del atleta; de los entrenadores que posean un salario mayor o igual a 2500; que los atletas hayan nacido a partir del año 1979 y que la calificación obtenida sea mayor al promedio general de notas asignadas. La forma de resolver esta consulta es dividirla en partes. La primer parte sería

Figura 100. Páginas 89 – 92 Libro “Bases de datos principiantes”

devolver todos los atletas que han nacido a partir de 1979. La siguiente parte sería devolver todos los entrenadores cuyo salario es mayor a 2500. La última parte sería devolver el promedio general de las notas. Estas partes se unirán en la consulta principal. La decisión de en qué cláusula se utilizará cada una queda a discreción. Este ejemplo utilizará las primeras dos partes en el *FROM* y la parte del promedio en el *WHERE*. La figura 3.24. muestra esta consulta.

Figura 3.24. Script ejemplo de subconsulta compleja

```
SELECT tabati.nombreati AS nombre_atleta, tabent.nombreent AS nombre_entrenador,
asi2.calificacion AS calificacion_atleta
FROM asignacion asi2,
(
   SELECT ati1.codigoati AS codigoati, ati1.nombreati AS nombreati,
   ati1.fecha_nacati as fecha_nac
   FROM atleta ati1
   WHERE ati1.fecha_nacati >= '01/01/1979'
)tabati,
(
   SELECT ent1.codigoent AS codigoent, ent1.nombreent AS nombreent,
   ent1.salario AS salario
   FROM entrenador ent1
   WHERE ent1.salario >=2500
)tabent
WHERE tabati.codigoati = asi2.codigoati
AND tabent.codigoent = asi2.codigoent
AND asi2.calificacion > (
                  SELECT AVG (asi1.calificacion)
                  FROM asignacion asi1
                  )
;
```

HOMBRE_ATLETA	HOMBRE_ENTRENADOR	CALIFICACION_ATLETA
D	James Lopez	90
F	Fernando Ruiz	85
E	James Lopez	85
D	Luisa Martinez	90

3.3.11. Subconsultas Anidadas

El concepto de subconsultas anidadas quiere decir que una subconsulta contendrá otra o más subconsultas dentro de ella y así sucesivamente. Esto ocasiona que existan consultas de n niveles. Las consultas de este tipo es común verlas, especialmente las consultas de dos niveles.

Ejemplo 3.24.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que liste los entrenadores cuyo salario es menor o igual al entrenador que tiene el menor salario que sobrepasa el promedio general de salarios. Esta consulta necesita que se encuentre el promedio general de salarios, posteriormente el salario que sobrepasa este valor y que más se acerca al mismo y por último listar todos los salarios inferiores a este valor. La figura 3.25. muestra esta consulta.

Figura 3.25. Script ejemplo subconsulta anidada

```
SELECT ent1.codigoent AS entrenador, ent1.nombreent AS nombre,
ent1.salario AS salario
FROM entrenador ent1
WHERE ent1.salario > (
  SELECT MIN (ent2.salario) AS min_sal
  FROM entrenador ent2
  WHERE ent2.salario > (
     SELECT AVG (ent3.salario) AS prom_gen_sal
     FROM entrenador ent3
  )
)
;
```

ENTRENADOR	NOMBRE	SALARIO
1	Juan Rosito Guttelewitz	1500
3	Gabino Ganza	1200,55
5	Carlos Gomez	1800

3.3.12. Ordenamientos

Los ordenamientos en SQL permiten ordenar los datos que se encuentran dentro de un *SELECT*.

La función *DISTINCT* sirve para poder hacer que en los datos que devuelve un *SELECT*, si hay datos repetidos (conjunto de campos) sean omitidos y solo se muestre una vez. Esto permite realizar búsquedas específicas, donde los valores repetidos son un estorbo.

La instrucción *ORDER BY* permite ordenar los valores obtenidos de una consulta por el campo que se desee. Esto siempre y cuando aparezca el mismo en el *SELECT*. Los ordenamientos se pueden utilizar en cascada (por niveles). Esto permite que los resultados de las consultas se muestren en una estructura

tipo árbol jerárquico horizontal. Esta instrucción se utiliza al final de la consulta. Esta instrucción puede utilizar ya sea el nombre de los campos, el alias de los campos del *SELECT* ó utilizar posiciones de los campos en el *SELECT* (1,2,3,…). La instrucción *ORDER BY* luego del nombre de cada campo puede tener una instrucción extra (*ASC*, *DESC*), esto permite el ordenamiento de ese campo ascendente o descendentemente.

Ejemplo 3.25.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva el nombre de los atletas que están asignados para entrenar las disciplinas. La consulta se realizará consultando la tabla asignación y a partir de ahí sacar todos los atletas que han practicado alguna disciplina. La figura 3.26. muestra esta consulta.

Figura 3.26. Script de consulta de ejemplo de *distinct*

```
SELECT DISTINCT ati.nombreati
FROM atleta ati, asignacion asi
WHERE ati.codigoati = asi.codigoati
;
```

NOMBREATL
D
A
B
C
E
F
G

Ejemplo 3.26.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que despliegue el nombre del atleta, el entrenador y la mayor calificación obtenida con ese entrenador, ordenados con el atleta ascendentemente, el entrenador descendente y la mayor calificación ascendentemente. La figura 3.27. muestra esta consulta.

Figura 3.27. Script ejemplo consulta ordenamiento

```
SELECT ati.nombreati AS atleta, ent.nombreent AS entrenador,
califmax.calificacion AS max_calificacion
FROM atleta ati, entrenador ent,
(
   SELECT asi2.codigoati AS codigoati, asi2.codigoent AS codigoent,
MAX(asi2.calificacion) AS calificacion
```



```
   FROM asignacion asi2
   GROUP BY asi2.codigoati, asi2.codigoent
) califmax
WHERE ati.codigoati = califmax.codigoati
AND ent.codigoent = califmax.codigoent
ORDER BY 1 ASC, ent.nombreent DESC, califmax.calificacion ASC
;
```

ATLETA	ENTRENADOR	MAX_CALIFICACION
A	Juan Rosito Guttelewitz	85
A	James Lopez	95
A	Gabino Ganza	92
B	Juan Rosito Guttelewitz	60
B	James Lopez	95
B	Gabino Ganza	37
C	Luisa Martinez	85
C	Juan Rosito Guttelewitz	85
C	Gabino Ganza	90
D	Luisa Martinez	90
D	James Lopez	90
E	Juan Rosito Guttelewitz	60
E	James Lopez	85
E	Fernando Ruiz	73
F	Fernando Ruiz	85
F	Carlos Gomez	90
G	Carlos Gomez	90

3.3.13. *HAVING*

La cláusula *HAVING* permite añadir una restricción al momento que se utiliza una función (*AVG, MAX, MIN, SUM, COUNT*). Esta restricción permitirá ahorrarse utilizar una subconsulta que realice alguna condición con respecto a la función. Esta cláusula se utiliza al final del *SELECT* que contiene la función y se coloca después del *GROUP BY*.

Ejemplo 3.27.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre el promedio de notas para los atletas. Los únicos promedios que se tomarán en cuenta son los que sean iguales o mayores a 85. La figura 3.28. muestra esta consulta.

Figura 3.28. Script consulta de ejemplo de *having*

Figura 101. Páginas 93 – 96 Libro “Bases de datos principiantes”


```
SELECT atl.nombreatl AS atleta, AVG(asi.calificacion) AS promedio
FROM atleta atl, asignacion asi
WHERE atl.codigoatl = asi.codigoatl
GROUP BY atl.nombreatl
HAVING AVG(asi.calificacion) >= 85
;
```

ATLETA	PROMEDIO
D	90
A	87
F	87.5

3.3.14. *IN* y *NOT IN*

IN es una función que es utilizada para realizar comparaciones con una serie de valores. Esto significa que de un lado del *IN* se coloca un valor y del otro un conjunto de valores y esta devuelve verdadero si el valor coincide dentro del grupo. El conjunto de valores puede ser colocado estáticamente listando los valores dentro de paréntesis. Estos también pueden ser el resultado de una subconsulta. El *NOT IN* es la negación del *IN*. Esto quiere decir que si el valor es encontrado dentro del conjunto de datos por utilizar la palabra *NOT* el valor es negado. Esto sería verdadero si el valor no se encuentra en los datos.

Ejemplo 3.28.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que liste los entrenadores que poseen un salario de 1000, 1500 ó 3000. La figura 3.29. muestra esta consulta.

Figura 3.29. Script consulta de *in* con valores estáticos

```
SELECT ent.codigoent AS entrenador, ent.nombreent AS nombre,
ent.salario AS salario
FROM entrenador ent
WHERE ent.salario IN (1000, 1500, 3000)
;
```

ENTRENADOR	NOMBRE	SALARIO
1	Juan Rosito Guttelewitz	1500
6	Luisa Martinez	3000
7	Mario Juarez	3000

Ejemplo 3.29.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que liste los entrenadores que no poseen un salario igual al listado de todos los salarios de los entrenadores más un bono de 1000. Esta consulta se basará en el uso de una subconsulta que devuelva el conjunto de salarios más la bonificación. Esta se comparará en la consulta principal y si el valor no se encuentra en el conjunto de datos será mostrado. La figura 3.30. muestra esta consulta.

Figura 3.30. Script consulta de not *in* con subconsulta

```
SELECT ent.codigoent AS entrenador, ent.nombreent AS nombre,
ent.salario AS salario
FROM entrenador ent
WHERE ent.salario NOT IN (
    SELECT ent2.salario + 1000
    FROM entrenador ent2
)
;
```

ENTRENADOR	NOMBRE	SALARIO
7	Mario Juarez	3000
6	Luisa Martinez	3000
5	Carlos Gomez	1800
2	James Lopez	2505
1	Juan Rosito Guttelewitz	1500
3	Gabino Gainza	1200,55

3.3.15. *BETWEEN*

La instrucción BETWEEN es similar a la instrucción IN. La diferencia entre ambas radica que BETWEEN comparará un rango de valores. Esto difiere de la instrucción IN que compara una ocurrencia comparando cada dato del conjunto de datos. La instrucción BETWEEN se limita a utilizar donde inicia y donde finaliza el rango de valores a verificar. Esta instrucción también acepta el NOT. Esto permite indicar si el valor a comparar no se encuentra dentro de ese rango.

Ejemplo 3.30.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que despliegue los salarios comprendidos entre 1000 y 2000. La figura 3.31. muestra esta consulta.

Figura 3.31. Script ejemplo de consulta con *between*

```
SELECT ent.codigoent AS entrenador, ent.nombreent AS nombre, ent.salario AS salario
FROM entrenador ent
WHERE ent.salario BETWEEN 1000 AND 2000
;
```

ENTRENADOR	NOMBRE	SALARIO
1	Juan Rosito Guttelewitz	1500
3	Gabino Gainza	1200,55
5	Carlos Gomez	1800

Ejemplo 3.31.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que despliegue los entrenadores que no poseen un salario dentro del rango La figura 3.32..

Figura 3.32. Script ejemplo de *not between*

```
SELECT ent.codigoent AS entrenador, ent.nombreent AS nombre,
ent.salario AS salario
FROM entrenador ent
WHERE ent.salario NOT BETWEEN 1000 AND 2000
;
```

ENTRENADOR	NOMBRE	SALARIO
2	James Lopez	2505
4	Fernando Ruiz	2500
6	Luisa Martinez	3000
7	Mario Juarez	3000

3.3.16. *EXISTS*

La cláusula *EXISTS* funciona con valores booleanos. Estos son valores que indican si existe o no existe (verdadero o falso). Esta cláusula funciona de esta manera: si la consulta usada en el *EXISTS* devuelve al menos una fila el valor es verdadero; si la consulta usada en el *EXISTS* no devuelve ninguna fila el valor es falso. La consulta en la cual se aplica esta cláusula debe entenderse bien debido a que al usar la cláusula puede generar confusión. La cláusula *EXISTS* se coloca dentro del *WHERE* y no necesita de ningún operador de

evaluación debido a que este devuelve verdadero o falso. Esta cláusula va acompañada siempre de una subconsulta.

Ejemplo 3.32.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva el nombre de los atletas que no tuvieron años de práctica, donde no tuvieron nivel de élite (calificación mayor a 80). La consulta debe ser escrita de una forma que se entienda. Esto es lo primero que se debe de hacer. La consulta quedaría de la siguiente forma: listar el nombre de los atletas en la cual todos sus años de práctica siempre han obtenido nivel de élite. Esto significa que si un atleta en alguna ocasión obtuvo una nota menor al nivel de élite, es totalmente rechazado. Esta consulta necesita de la negación para poder realizarse. Esto es debido a que si se encuentra al menos una nota menor al nivel de élite será devuelta en el select. Esto origina que al negar aunque haya un registro devuelto, esto hace falsa la condición y el atleta es omitido debido a que posee una nota menor al nivel de élite y se procede a encontrar un atleta que siempre haya tenido notas de nivel de élite. La figura 3.33. muestra esta consulta.

Figura 3.33. Script de ejemplo de consulta usando EXISTS 1

```
SELECT atl.codigoatl as atleta, atl.nombreatl as nombre_atleta
FROM atleta atl
WHERE NOT EXISTS (
    SELECT asi.codigoatl
    FROM asignacion asi
    WHERE atl.codigoatl = asi.codigoatl
    AND asi.calificacion < 81
)
;
```

ATLETA	NOMBRE_ATLETA
4	D
6	F

Ejemplo 3.33.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que encuentre a los atletas que llegaron a nivel de élite en la disciplina Carrera. Esta consulta se realizará utilizando subconsultas. Esta consulta se basará utilizando dos *EXISTS*. El *EXISTS* interior permitirá encontrar si un atleta se ha asignado una disciplina con el nombre ecuestre. El *EXISTS* externo se utilizará para verificar si ya que sabemos que el atleta se asigno una disciplina ecuestre, en ésta obtuvo una nota de élite. La consulta principal mostrará al atleta. Esta consulta con un registro que encuentre que

Figura 102. Páginas 97 – 100 Libro “Bases de datos principiantes”

posee nota de élite en la disciplina carrera devolverá un valor. Esta consulta permite mostrar el uso de *EXISTS* en cascada y anidados. La figura 3.34. muestra esta consulta.

Figura 3.34. Script de ejemplo de consulta usando *EXISTS* 2

```
SELECT atl.codigoatl as atleta, atl.nombreatl as nombre_atleta
FROM atleta atl
WHERE EXISTS (
    SELECT asi.codigoatl
    FROM asignacion asi
    WHERE atl.codigoatl = asi.codigoatl
    AND asi.calificacion >= 80
    AND
      EXISTS (
      SELECT dis.codigodis
      FROM disciplina dis
      WHERE dis.codigodis = asi.codigodis
      AND dis.nombredis = 'Carrera'
    )
)
;
```

ATLETA	NOMBRE_ATLETA
1	A
6	F
5	E
3	C

Ejemplo 3.34.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva el nombre de atletas que han recibido entrenamiento con los entrenadores que han entrenado al atleta C. La consulta debe verificar que exista un atleta en la consulta que devuelve los compañeros del atleta C. Esto posteriormente debe verificar que no se esté verificando el mismo atleta C. Esto origina que se obtengan todos los compañeros de C. La figura 3.35..

Figura 3.35. Script ejemplo consulta usando *EXISTS* 3

```
SELECT DISTINCT at2.nombreatl as compañeros_d_C
FROM asignacion as2, atleta at2
WHERE
    as2.codigoatl = at2.codigoatl
    AND at2.nombreatl <> 'C'
    AND EXISTS (
                SELECT as3.codigoent
                FROM asignacion as3, atleta at3
                WHERE  at3.nombreatl = 'C'
                AND as3.codigoatl = at3.codigoatl
                AND as3.codigoent = as2.codigoent
    )
```



```
ORDER BY 1
;
```

COMPAÑEROS_D_C
A
B
D
E

Ejemplo 3.35.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva el nombre de todos los atletas que han sido entrenados por el entrenador llamado James Lopez. Esta consulta debe contener una subconsulta que devuelva si un atleta ha entrenado con James Lopez. Esto permite que posteriormente se verifique si un atleta ha sido asignado a ese entrenador. La figura 3.36. muestra esta consulta.

Figura 3.36. Script ejemplo consulta usando *EXISTS* 4

```
SELECT atl.codigoatl as atleta, atl.nombreatl as nombre
FROM atleta atl
WHERE EXISTS (
  SELECT asi.codigoatl
  FROM asignacion asi
  WHERE atl.codigoatl = asi.codigoatl
  AND EXISTS (
        SELECT ent.codigoent
        FROM entrenador ent
        WHERE ent.nombreent = 'James Lopez'
        AND ent.codigoent = asi.codigoent
    )
)
;
```

ATLETA	NOMBRE
1	A
2	B
4	D
5	E

3.3.17. *UNION*

EL operador *UNION* permite unir dos consultas. Este operador es de dos formas *UNION ALL* y *UNION*. La diferencia entre ambas es que *UNION ALL* une ambas consultas sin importar si hay datos repetidos. *UNION* a diferencia permite unir las dos consultas pero los datos repetidos son puestos una sola vez, algo asi como si se estuviera colocando *DISTINCT*. La única consideración del uso de esta cláusula es que ambas consultas deben poseer la misma cantidad de campos y del mismo tipo en el mismo orden. No hay límite de consultas que se pueden unir.

Ejemplo 3.36.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre los entrenadores que ganan menos de 3000 y los entrenadores que ganan menos de 2000. Esta consulta se puede realizar con restricciones pero para motivos de ejemplo utilizaremos UNION y UNION ALL. La primer consulta será la de los entrenadores que ganan menos a 2000, la otra consulta corresponderá a los entrenadores que ganan menos de 2000. La figura 3.37. muestra ambas consultas y sus resultados.

Figura 3.37. Script consulta ejemplo de *UNIONS*

```
SELECT ent.codigoent as entrenador, ent.nombreent as nombre
FROM entrenador ent
WHERE ent.salario < 3000
UNION ALL
SELECT ent.codigoent, ent.nombreent
FROM entrenador ent
WHERE ent.salario < 2000
;
```

ENTRENADOR	NOMBRE
1	Juan Rosito Guttelewitz
2	James Lopez
3	Gabino Gainza
4	Fernando Ruiz
5	Carlos Gomez
1	Juan Rosito Guttelewitz
3	Gabino Gainza
5	Carlos Gomez

```
SELECT ent.codigoent as entrenador, ent.nombreent as nombre
FROM entrenador ent
WHERE ent.salario < 3000
UNION
SELECT ent.codigoent, ent.nombreent
FROM entrenador ent
```



```
WHERE ent.salario < 2000
;
```

ENTRENADOR	NOMBRE
1	Juan Rosito Guttelewitz
2	James Lopez
3	Gabino Gainza
4	Fernando Ruiz
5	Carlos Gomez

3.3.18. *MINUS*

El operador *MINUS* toma los valores de la primera consulta y le quita los valores que están en la segunda consulta. Esto quiere decir que solamente mostrará los valores que no existen en la segunda consulta. El operador *MINUS* se utiliza cuando es más fácil crear dos consultas separadas que posteriormente se pueden juntar.

Ejemplo 3.37.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre los entrenadores que poseen un salario comprendido entre 1500 y 2000, sin incluir el 2000. Esta consulta se puede realizar ya sea utilizando *BETWEEN* ó restricciones. El ejemplo será realizado utilizando la cláusula *MINUS*. La consulta necesita de dos consultas separadas, una de ellas que devuelva los salarios menores a 2000 y la otra menores a 1500. Lo siguiente será utilizar *MINUS* para mostrar este intervalo. La figura 3.38. muestra esta consulta.

Figura 3.38. Script ejemplo de consulta utilizando *minus*

```
SELECT ent.codigoent as entrenador, ent.nombreent as nombre, ent.salario
FROM entrenador ent
WHERE ent.salario < 2000
MINUS
SELECT ent.codigoent, ent.nombreent, ent.salario
FROM entrenador ent
WHERE ent.salario < 1500
;
```

ENTRENADOR	NOMBRE	SALARIO
1	Juan Rosito Guttelewitz	1500
5	Carlos Gomez	1800

Figura 103. Páginas 101 – 104 Libro “Bases de datos principiantes”

3.3.19. *INSTERSECT*

El operador *INTERSECT* permite devolver los valores que se encuentran en ambas consultas. Esto quiere decir que lo que aparece en la consulta 1 y lo que también aparece en la consulta 2 serán devueltos. Esto significa lo que hay en la consulta 1 que también hay en la consulta 2.

Ejemplo 3.38.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que muestre los entrenadores que aparecen tanto en los salarios menores a 2000 y a los entrenadores con salarios mayores a 1000. Esta consulta se realizará con dos consultas unidas por *INTERSECT*. La figura 3.39. .

Figura 3.39. Script

```
SELECT ent.codigoent as entrenador, ent.nombreent as nombre, ent.salario
FROM entrenador ent
WHERE ent.salario < 2000
INTERSECT
SELECT ent.codigoent, ent.nombreent, ent.salario
FROM entrenador ent
WHERE ent.salario > 1000
;
```

ENTRENADOR	NOMBRE	SALARIO
1	Juan Rosito Guttelewitz	1500
3	Gabino Gainza	1200,55
5	Carlos Gomez	1800

3.3.20. *LIKE*

El operador LIKE permite realizar una búsqueda que utiliza un patrón en lugar de especificar o dar un rango de lo que se desea buscar. Este operador trabaja sobre cadenas. Esta utiliza los caracteres: _, %. El carácter de _ permite indicar que se omitirá un carácter en la cadena hasta llegar al patrón. El carácter % permite indicar que se omitirán todos los caracteres hasta que se obtenga el patrón deseado. Esto permite que se puedan combinar éstos para

obtener una búsqueda acorde a lo deseado. Existen otros operadores y funciones que trabajan sobre cadenas. Estos dependerán del DBMS que se esté utilizando.

Ejemplo 3.39.

A partir de las tablas del ejemplo 3.1. que se encuentran en la figura 3.2., realizar una consulta que devuelva todos los entrenadores cuyo nombre empiece con F. Los entrenadores cuyo apellido empiece con G. Los entrenadores cuya segunda letra del nombre empiece con a. Esta consulta bastara con añadir tres restricciones utilizando LIKE. Esto permitirá listar los nombres pero además necesitamos que se conecten por OR debido a que se solicita cualquiera de las ocurrencias para listar un entrenador. La figura 3.40. muestra esta consulta.

Figura 3.40. Script

```
SELECT DISTINCT ent.nombreent AS entrenador
FROM entrenador ent
WHERE ent.nombreent LIKE 'F%'
OR ent.nombreent LIKE '%G%'
OR ent.nombreent LIKE '_a%'
ORDER BY 1
;
```

ENTRENADOR
Carlos Gomez
Fernando Ruiz
Gabino Gainza
James Lopez
Juan Rosito Guttelewitz
Mario Juarez

3.4. Ejemplo Práctico de SQL

El ejemplo práctico de SQL se basa en la sección 2.4., específicamente en el enunciado del problema que se describe en la sección 2.4.1. Este también se basa en el modelo entidad relación que se encuentra en la figura 2.47. El problema adquiere su estructura a partir de las tablas 2.31 a 2.42.

La primer parte de este ejemplo práctico de SQL será realizar el esquema conceptual del modelo de la figura 2.47. La segunda parte consistirá en realizar las consultas partiendo del modelo de la figura 2.47.

3.4.1. Esquema Conceptual

El esquema conceptual se realizará creando las tablas que se encuentran mapeadas en las tablas 2.31 a 2.42. Estas se encuentran en la sección 2.4.2.2. de la sección mapeo conceptual. La figura 3.42. muestra el script de creación de las tablas.

Figura 3.42. Script de creación de las tablas del problema de elecciones

```
CREATE TABLE TIPOLUGAR
(
  idtipolugar NUMERIC NOT NULL,
  nombtipolugar VARCHAR2(50) NOT NULL,
  CONSTRAINT pk_idtipolugar PRIMARY KEY (idtipolugar)
);

CREATE TABLE LUGAR
(
  idlugar1 NUMERIC NOT NULL,
  nombLugar VARCHAR2(50) NOT NULL,
  descLugar NUMERIC NOT NULL,
  idlugar2 NUMERIC NOT NULL,
  idtipolugar NUMERIC NOT NULL,
  CONSTRAINT fk_lugar_idtipolugar FOREIGN KEY (idtipolugar) REFERENCES TIPOLUGAR(idtipolugar),
  CONSTRAINT fk_lugar_idlugar1 FOREIGN KEY (idlugar2) REFERENCES LUGAR(idlugar1),
  CONSTRAINT pk_idlugar1 PRIMARY KEY (idlugar1)
);

CREATE TABLE TIPOELECCION
(
  idtipoeleccion NUMERIC NOT NULL,
  nombtipoeleccion VARCHAR2(50) NOT NULL,
  CONSTRAINT pk_idtipoeleccion PRIMARY KEY (idtipoeleccion)
);

CREATE TABLE ELECCION
(
  ideleccion NUMERIC NOT NULL,
  nombeleccion VARCHAR2(50) NOT NULL,
  anio DATE NOT NULL,
  periodoini DATE NULL,
  periodofin DATE NULL,
  idtipoeleccion NUMERIC NOT NULL,
  idlugar1 NUMERIC NOT NULL,
  CONSTRAINT fk_eleccion_idtipoeleccion FOREIGN KEY (idtipoeleccion) REFERENCES
TIPOELECCION(idtipoeleccion),
  CONSTRAINT fk_eleccion_idlugar1 FOREIGN KEY (idlugar1) REFERENCES LUGAR(idlugar1),
  CONSTRAINT pk_ideleccion PRIMARY KEY (ideleccion)
);

CREATE TABLE TIPOPUESTO
(
  idtipopuesto NUMERIC NOT NULL,
  nombtipopuesto VARCHAR2(50) NOT NULL,
  CONSTRAINT pk_idtipopuesto PRIMARY KEY (idtipopuesto)
);
```



```
CREATE TABLE TIPOPOLITICO
(
  idtipopolitico NUMERIC NOT NULL,
  nombtipopolitico VARCHAR2(50) NOT NULL,
  CONSTRAINT pk_idtipopolitico PRIMARY KEY (idtipopolitico)
);

CREATE TABLE POLITICO
(
  idpolitico NUMERIC NOT NULL,
  nombpolitico VARCHAR2(50) NOT NULL,
  despolitico VARCHAR2(50) NULL,
  idtipopolitico NUMERIC NOT NULL,
  CONSTRAINT fk_politico_idtipopolitico FOREIGN KEY (idtipopolitico) REFERENCES TIPOPOLITICO(idtipopolitico),
  CONSTRAINT pk_idpolitico PRIMARY KEY (idpolitico)
);

CREATE TABLE PUESTO
(
  idpuesto NUMERIC NOT NULL,
  nombpuesto VARCHAR2(50) NOT NULL,
  despuesto VARCHAR2(50) NULL,
  idtipopuesto NUMERIC NOT NULL,
  CONSTRAINT fk_puesto_idtipopuesto FOREIGN KEY (idtipopuesto) REFERENCES TIPOPUESTO(idtipopuesto),
  CONSTRAINT pk_idpuesto PRIMARY KEY (idpuesto)
);

CREATE TABLE POBLACION
(
  idpoblacion NUMERIC NOT NULL,
  descpoblacion VARCHAR2(50) NULL,
  cantpoblacion NUMERIC NOT NULL,
  CONSTRAINT pk_idpoblacion PRIMARY KEY (idpoblacion)
);

CREATE TABLE ATRIBUTO
(
  idatributo NUMERIC NOT NULL,
  nombattributo VARCHAR2(50) NOT NULL,
  tipoatributo NUMERIC NOT NULL,
  CONSTRAINT pk_idatributo PRIMARY KEY (idatributo)
);

CREATE TABLE CARACTERISTICA
(
  valorchar VARCHAR2(50) NOT NULL,
  valornum NUMERIC NOT NULL,
  valorfdate DATE NOT NULL,
  idpoblacion NUMERIC NOT NULL,
  idatributo NUMERIC NOT NULL,
  CONSTRAINT fk_caracpoblacion FOREIGN KEY (idpoblacion) REFERENCES POBLACION(idpoblacion),
  CONSTRAINT fk_caracidatributo FOREIGN KEY (idatributo) REFERENCES ATRIBUTO(idatributo),
  CONSTRAINT pk_caracteristica PRIMARY KEY (idpoblacion, idatributo)
);

CREATE TABLE VOTACION
(
  idpoblacion NUMERIC NOT NULL,
  idpuesto NUMERIC NOT NULL,
  ideleccion NUMERIC NOT NULL,
  idpolitico NUMERIC NOT NULL,
  fechavota DATE NOT NULL,
  observacion VARCHAR2(50) NOT NULL,
  CONSTRAINT fk_vota_idpolitico FOREIGN KEY (idpolitico) REFERENCES POLITICO(idpolitico),
  CONSTRAINT fk_vota_ideleccion FOREIGN KEY (ideleccion) REFERENCES ELECCION(ideleccion),
```

Figura 104. Páginas 105 – 108 Libro “Bases de datos principiantes”


```
  CONSTRAINT fk_vota_idpoblacion FOREIGN KEY (idpoblacion) REFERENCES POBLACION(idpoblacion),
  CONSTRAINT fk_vota_idpuesto FOREIGN KEY (idpuesto) REFERENCES PUESTO(idpuesto),
  CONSTRAINT pk_votacion PRIMARY KEY (idpoblacion,idpuesto,idpolitico,ideleccion,fechavota)
);
```

3.4.2. Consultas

El Instituto Centroamericano Electoral ha realizado una serie de veinte consultas que necesitan que el sistema electoral sea capaz de responder. Estas consultas serán realizadas a partir del modelo electoral realizado en la figura 2.47. y cuyo script de creación se encuentra en la figura 3.42. Esta sección brindará un detalle del enunciado de cada consulta y su solución en lenguaje SQL, asimismo de una imagen que muestre como será devuelto el resultado de la consulta.

3.4.2.1. Consulta No. 1

Desplegar para cada elección: el pais y el partido politico que obtuvo mayor porcentaje de votos en su pais. Debe desplegarse el nombre de la elección, el año de la elección, el pais, el nombre del partido politico y el porcentaje que obtuvo de votos en su pais.

La consulta se realizará utilizando dos tipos de consultas de *from*. Estas subconsultas serán en su estructura semejantes. Estas se diferenciarán debido a que la primera encontrará únicamente el porcentaje que obtiene cada partido politico, en una elección, en un año, en un pais. La otra subconsulta encontrará la máxima nota del porcentaje obtenido en una elección, en un año, en un pais. La consulta principal permitirá comparar el máximo porcentaje de la subconsulta 2 con todos los porcentajes de la consulta 1 y de esta forma listar todos los partidos politicos que obtuvieron el mayor porcentaje de votos en un pais, para una elección en especifico, en un año.

El año se encuentra utilizando la función *to_char*. Esta función permite entre sus muchas funciones extraer el año de una fecha y convertirlo a tipo carácter.

Las subconsultas deben utilizar otra subconsulta anidada cada una. Esta debido al nivel de abstracción del modelo es necesaria para poder mostrar un listado de los paises, regiones, departamentos y municipios. Esto facilita el poder realizar comparaciones y cálculos. La figura 3.43. muestra el script de esta consulta.

Figura 3.43. Script consulta 1

```
SELECT tabsinmax.npais as pais, tabsinmax.nombpolitico as partido_politico,
       tabsinmax.nombeleccion as eleccion , to_char(tabsinmax.anio, 'yyyy') as año,
       tabsinmax.porcentaje as porcentaje_votos
FROM
(
        SELECT tabtotalpartido.idpais, tabtotalpartido.npais, tabtotalpartido.idpolitico,
               tabtotalpartido.nombpolitico,
               tabtotalpartido.ideleccion, tabtotalpartido.nombeleccion, tabtotalpartido.anio,
               ROUND((tabtotalpartido.totalvotosp*100)/tabtotalele.totalvotos),2) as porcentaje
        FROM
        (
                SELECT ele1.ideleccion, SUM(vota1.cantvoto) as totalvotos
                FROM eleccion ele1, votacion vota1
                WHERE ele1.ideleccion = vota1.ideleccion
                GROUP BY ele1.ideleccion
        ) tabtotalele,
        (
                SELECT tablugar.idpais, tablugar.npais, pol.idpolitico, pol.nombpolitico,
                ele.ideleccion, ele.nombeleccion, ele.anio, SUM(vota.cantvoto) as totalvotosp
                FROM tipoeleccion tele, eleccion ele, tipopolitico tpol, politico pol, votacion vota,
                (
                        SELECT  DISTINCT lu1.idlugar1 as idpais, lu2.idlugar1 as idregion,
                                                                 lu3.idlugar1 as iddepto, lu4.idlugar1 as idmuni,
                                                                 lu1.nomblugar as npais, lu2.nomblugar as
nregion,
                                                                 lu3.nomblugar as ndepto, lu4.nomblugar as
nmuni
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )  tablugar
                WHERE  tele.nombtipoeleccion like '%Municipal%' AND
                                        tele.idtipoeleccion = ele.idtipoeleccion AND
                                        ele.idlugar1 = tablugar.idmuni AND
                                        tpol.idtipopolitico = pol.idtipopolitico AND
                                        ele.ideleccion = vota.ideleccion AND
                                        pol.idpolitico = vota.idpolitico
                GROUP BY tablugar.idpais, tablugar.npais, pol.idpolitico, pol.nombpolitico,
                                        ele.ideleccion, ele.nombeleccion, ele.anio
        ) tabtotalpartido
        WHERE tabtotalele.ideleccion = tabtotalpartido.ideleccion
) tabsinmax,
(
        SELECT tabtotalpartido.idpais, tabtotalpartido.ideleccion,
               ROUND(MAX(((tabtotalpartido.totalvotosp*100)/tabtotalele.totalvotos)),2) as maxporcentaje
        FROM
        (
                SELECT ele1.ideleccion, SUM(vota1.cantvoto) as totalvotos
                FROM eleccion ele1, votacion vota1
                WHERE ele1.ideleccion = vota1.ideleccion
                GROUP BY ele1.ideleccion
```



```
        ) tabtotalele,
        (
                SELECT tablugar.idpais, tablugar.npais, pol.idpolitico, pol.nombpolitico,
                ele.ideleccion, ele.nombeleccion, SUM(vota.cantvoto) as totalvotosp
                FROM tipoeleccion tele, eleccion ele, tipopolitico tpol, politico pol, votacion vota,
                (
                        SELECT  DISTINCT lu1.idlugar1 as idpais, lu2.idlugar1 as idregion,
                                                                 lu3.idlugar1 as iddepto, lu4.idlugar1 as idmuni,
                                                                 lu1.nomblugar as npais, lu2.nomblugar as
nregion,
                                                                 lu3.nomblugar as ndepto, lu4.nomblugar as
nmuni
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )  tablugar
                WHERE  tele.nombtipoeleccion like '%Municipal%' AND
                                        tele.idtipoeleccion = ele.idtipoeleccion AND
                                        ele.idlugar1 = tablugar.idmuni AND
                                        tpol.idtipopolitico = pol.idtipopolitico AND
                                        ele.ideleccion = vota.ideleccion AND
                                        pol.idpolitico = vota.idpolitico
                GROUP BY tablugar.idpais, tablugar.npais, pol.idpolitico, pol.nombpolitico,
                                        ele.ideleccion, ele.nombeleccion
        ) tabtotalpartido
        WHERE tabtotalele.ideleccion = tabtotalpartido.ideleccion
        GROUP BY tabtotalpartido.idpais, tabtotalpartido.ideleccion
) tabconmax
WHERE tabsinmax.ideleccion = tabconmax.ideleccion AND
    tabsinmax.porcentaje = tabconmax.maxporcentaje
;
```

	PAIS	PARTIDO_POLITICO	ELECCION	AÑO	PORCENTAJE_VOTOS
1	EL SALVADOR	Alianza Nac. Rep.	Elecciones Municipales	2005	39,07
2	EL SALVADOR	Alianza Nac. Rep.	Elecciones Municipales	2005	33,52
3	EL SALVADOR	Farabundo Martí lib. Nac.	Elecciones Municipales	2005	35,04
4	EL SALVADOR	Partido Accion Nacional	Elecciones Municipales	2005	35
5	EL SALVADOR	Partido Accion Nacional	Elecciones Municipales	2005	37,81
6	EL SALVADOR	Partido Accion Nacional	Elecciones Municipales	2005	36,78
7	EL SALVADOR	Partido Accion Nacional	Elecciones Municipales	2005	35,29
8	EL SALVADOR	Partido Accion Nacional	Elecciones Municipales	2005	36,33
9	EL SALVADOR	Partido Accion Nacional	Elecciones Municipales	2005	35,19
10	EL SALVADOR	Farabundo Martí lib. Nac.	Elecciones Municipales	2005	37,82

3.4.2.2. Consulta No. 2

Desplegar el total de votos y el porcentaje de votos emitidos por mujeres indigenas alfabetos.

La consulta principal consta de dos subconsultas de *from*. La primer subconsulta devuelve el total de votos emitidos por mujeres indigenas alfabetos. Esta subconsulta se encuentra formada por dos subconsultas más. Una subconsulta que indica que poblaciones son alfabetos (primaria, nivel medio, universitarios), la otra subconsulta devuelve que poblaciones son mujeres.

La segunda subconsulta de la consulta principal devuelve la cantidad de votos realizados en todas las elecciones. Esto permitirá que la consulta principal pueda encontrar el porcentaje y el total de votos para las mujeres indigenas alfabetos. La figura 3.44. muestra el script de esta consulta.

Figura 3.44. Script consulta 2

```
SELECT round((CANTMA1.cantmujalf5*100)/CANTVOT1.cantvotantes5,2)|' %' as
                "% MUJERES INDIGENAS ALFABETAS", CANTMA1.cantmujalf5 as "# VOTOS MUJ IND
ALFABETAS"
FROM
        (
        SELECT  SUM (vot5.cantvoto) AS cantmujalf5
        FROM
        (
                SELECT P.ALF5.idpoblacion5 as idpoblacion55
                FROM
                (
                        SELECT  DISTINCT pob5.idpoblacion as idpoblacion5
                        FROM CARACTERISTICA car5, ATRIBUTO atr5, POBLACION pob5,
                        (
                                SELECT  rta5.idatributo as rta5idatributo
                                FROM    ATRIBUTO rta5
                                WHERE  rta5.nombatributo like '%PRIMARIA%' OR
                                                                rta5.nombatributo like '%NIVEL MEDIO%' OR
                                                                rta5.nombatributo like '%UNIVERSITARIOS%'
                        )  ALFTAB5
                        WHERE  atr5.idatributo = ALFTAB5.rta5idatributo AND
                                                car5.idatributo = atr5.idatributo AND
                                                pob5.idpoblacion = car5.idpoblacion
                        ORDER BY pob5.idpoblacion
                ) P.ALF5,
                (
                        SELECT  pob5.idpoblacion as idpoblacion5
                        FROM CARACTERISTICA car5, ATRIBUTO atr5, POBLACION pob5
                        WHERE  atr5.nombatributo like '%mujeres%' AND
                                                car5.idatributo = atr5.idatributo AND
                                                pob5.idpoblacion = car5.idpoblacion
                ) MUJE5,
                (
                        SELECT  pob5.idpoblacion as idpoblacion5
                        FROM CARACTERISTICA car5, ATRIBUTO atr5, POBLACION pob5
                        WHERE  atr5.nombatributo like '%INDIGENAS%' AND
                                                car5.idatributo = atr5.idatributo AND
                                                pob5.idpoblacion = car5.idpoblacion
```

Figura 105. Páginas 109 – 112 Libro “Bases de datos principiantes”

```
                ) INDIG5
                WHERE PALF5.idpoblacion5 = MUJE5.idpoblacion5 AND
                      INDIG5.idpoblacion5 = MUJE5.idpoblacion5
        ) POBAM5,
        VOTACION vot5
        WHERE  POBAM5.idpoblacion55 = vot5.idpoblacion
        ) CANTMA1,
        (
                SELECT sum(vot1.cantvoto) as cantvotantes5
                FROM VOTACION vot1
        ) CANTVOT1
.
```

1 12,47 %	15700048

3.4.2.3. Consulta No. 3

Desplegar total de votos y porcentaje de votos de mujeres por departamento y pais. El ciento por ciento es el total de votos de mujeres por pais. (Tip: Todos los porcentajes por departamento de un pais deben sumar el 100%).

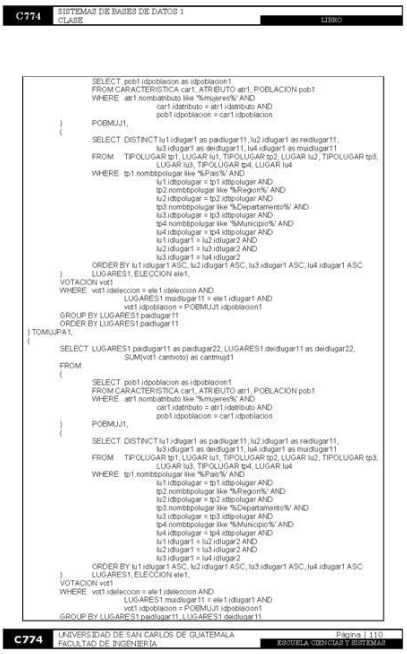
La consulta principal se encuentra integrada por dos subconsultas de from. La primer subconsulta devuelve el total de votos de mujeres por pais. Esta subconsulta se encuentra integrada por una subconsulta que devuelve las poblaciones mujeres y otra subconsulta que devuelve los paises donde votaron estas poblaciones mujeres.

La segunda subconsulta permite encontrar el total de votos de poblaciones mujeres por departamento. Esta subconsulta se integra por una subconsulta que encuentra que poblaciones son mujeres y otra que devuelve los departamentos donde han votado esas poblaciones mujeres.

La consulta principal entonces realiza el cálculo del porcentaje de las poblaciones mujeres por departamento tomando en cuenta los totales de las poblaciones mujeres de los paises. Esta consulta entonces devuelve el total de mujeres y su porcentaje de votos por departamentos. La figura 3.45. muestra el script de esta consulta.

Figura 3.45. Script consulta 3

```
SELECT lug1.nomblugar as "NOMBRE PAIS", lug2.nomblugar as "NOMBRE DEPARTAMENTO",
                round((TOMUJDE1.cantmujd1*100)/TOMUJPA1.totalmujp1,2) as "% MUJERES"
FROM
(
        SELECT LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalmujp1
        FROM
        (
```

```
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%mujeres%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )  POBMUJ1,
        (
                SELECT DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as mudlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        ) LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.mudlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBMUJ1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11
        ORDER BY LUGARES1.paidlugar11
) TOMUJPA1,
(
        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.deidlugar11 as deidlugar22,
                        SUM(vot1.cantvoto) as cantmujd1
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%mujeres%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )  POBMUJ1,
        (
                SELECT DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as mudlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        ) LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.mudlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBMUJ1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11, LUGARES1.deidlugar11
```


```
        ORDER BY LUGARES1.paidlugar11 ASC, LUGARES1.deidlugar11 ASC
) TOMUDE1, LUGAR lug1, LUGAR lug2
WHERE  TOMUDE1.paidlugar22 = TOMUPA1.paidlugar22 AND
                TOMUDE1.paidlugar22 = lug1.idlugar1 AND
                TOMUDE1.deidlugar22 = lug2.idlugar1
ORDER BY lug1.nomblugar, lug2.nomblugar, "% MUJERES" DESC
.
```

	NOMBRE PAIS	NOMBRE DEPARTAMENTO	% MUJERES
1	Costa Rica	Alajuela	25,12
2	Costa Rica	Cartago	12,43
3	Costa Rica	Guanacaste	18,92
4	Costa Rica	Heredia	16,11
5	Costa Rica	Limon	9,51
6	Costa Rica	Puntarenas	17,91
7	EL SALVAD...	Cabañas	3,62
8	EL SALVAD...	Chalatenango	13,34
9	EL SALVAD...	Cuscatlán	6,13
10	EL SALVAD...	La Libertad	8,57

3.4.2.4. Consulta No. 4

Desplegar el nombre del pais, el porcentaje de votos de ese pais en el que han votado mayor porcentaje de analfabetas. (tip: solo desplegar un nombre de pais, el de mayor porcentaje).

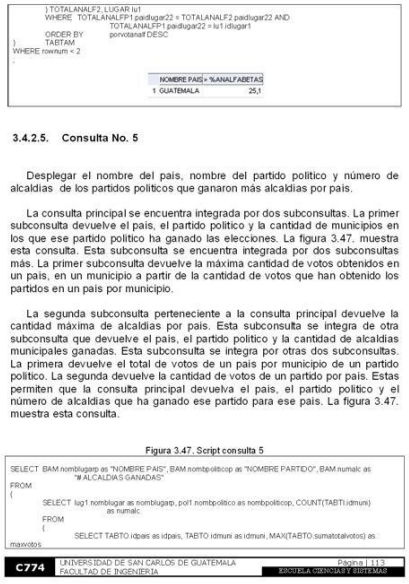
La consulta 4 posee una subconsulta que permite encontrar el pais y el porcentaje de votos de analfabetas. Esta subconsulta se encuentra integrada por dos subconsultas. La primer subconsulta encuentra las poblaciones analfabetas y realiza un cálculo que permite encontrar el total de votos de alfabetos que votaron por pais. La segunda subconsulta permite encontrar el total de votos realizados en un pais. Luego la subconsulta que comprende a estas dos realiza el cálculo del porcentaje de analfabetos que votaron por pais y realiza el ordenamiento descendente dejando al pais donde más votos existen en la posición 1. La consulta principal entonces listará únicamente la primera posición que comprenderá el pais con porcentaje donde votaron más analfabetas. La figura 3.46. muestra esta consulta.

Figura 3.46. Script de consulta 4

```
SELECT  TABTAM.nomblugar1 as "NOMBRE PAIS", TABTAM.porvotanalf as "> %ANALFABETAS"
FROM
(
```


```
SELECT lu1.nomblugar as nomblugar1, round((TOTALANALFP1.totalanalfpa1* 100)/
                TOTALANALF2.totalp1,2) AS porvotanalf
FROM
(
        SELECT LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalanalfpa1
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%ANALFABETOS%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )  POBANALF1,
        (
                SELECT DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as mudlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        ) LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.mudlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBANALF1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11
        ORDER BY LUGARES1.paidlugar11
) TOTALANALFP1,
(
        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalp1
        FROM    POBLACION pobla1,
        (
                SELECT DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as mudlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        ) LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.mudlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = pobla1.idpoblacion
        GROUP BY LUGARES1.paidlugar11
        ORDER BY LUGARES1.paidlugar11
```

Figura 106. Páginas 113 – 116 Libro “Bases de datos principiantes”




```
        ) TOTALANALF2, LUGAR lu1
        WHERE  TOTALANALFP1.paidlugar22 = TOTALANALF2.paidlugar22 AND
                TOTALANALFP1.paidlugar22 = lu1.idlugar1
        ORDER BY        porvotanalf DESC
)       TABTAM
WHERE rownum < 2
;
```

	NOMBRE PAIS	% ANALFABETAS
1	GUATEMALA	25,1

3.4.2.5. Consulta No. 5

Desplegar el nombre del pais, nombre del partido politico y número de alcaldias de los partidos politicos que ganaron más alcaldias por pais.

La consulta principal se encuentra integrada por dos subconsultas. La primer subconsulta devuelve el pais, el partido politico y la cantidad de municipios en los que ese partido politico ha ganado las elecciones. La figura 3.47. muestra esta consulta. Esta subconsulta se encuentra integrada por dos subconsultas más. La primer subconsulta devuelve la máxima cantidad de votos obtenidos en un pais, en un municipio a partir de la cantidad de votos que han obtenido los partidos en un pais por municipio.

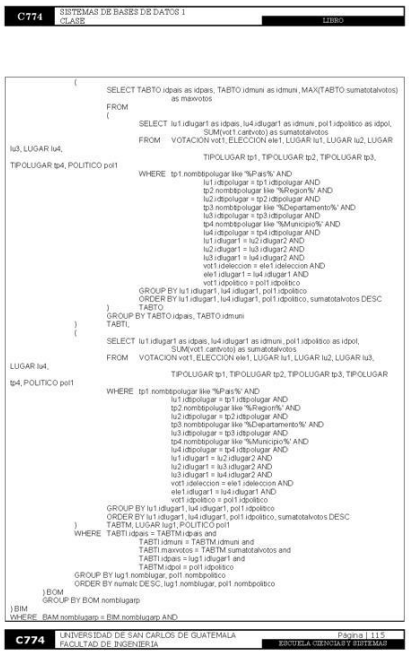
La segunda subconsulta perteneciente a la consulta principal devuelve la cantidad máxima de alcaldias por pais. Esta subconsulta se integra de otra subconsulta que devuelve el pais, el partido politico y la cantidad de alcaldias municipales ganadas. Esta subconsulta se integra por otras dos subconsultas. La primera devuelve el total de votos de un pais por municipio de un partido politico. La segunda devuelve la cantidad de votos de un partido por pais. Estas permiten que la consulta principal devuelva el pais, el partido politico y el número de alcaldias que ha ganado ese partido para ese pais. La figura 3.47. muestra esta consulta.

Figura 3.47. Script consulta 5

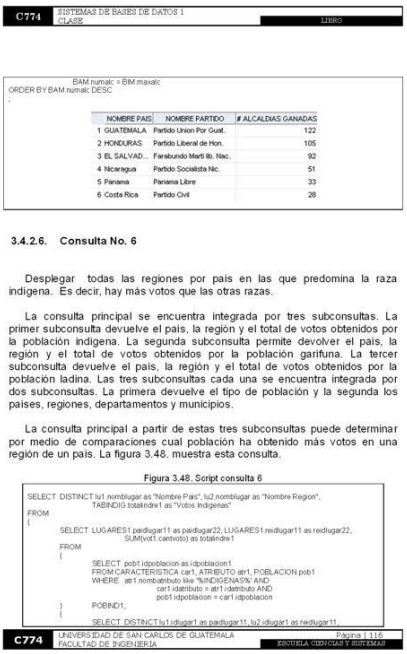
```
SELECT BAM.nomblugarp as "NOMBRE PAIS", BAM.nombpoliticop as "NOMBRE PARTIDO", BAM.numalc as
                "# ALCALDIAS GANADAS"
FROM
(
        SELECT lug1.nomblugar as nomblugarp, pol1.nombpolitico as nombpoliticop, COUNT(TABTI.idmuni)
                        as numalc
        FROM
        (
                SELECT TABTO.idpais as idpais, TABTO.idmuni as idmuni, MAX(TABTO.sumatotalvotos) as
maxvotos
```



```
                FROM
                (
                        SELECT  lu1.idlugar1 as idpais, lu4.idlugar1 as idmuni, pol1.idpolitico as idpol,
                                        SUM(vot1.cantvoto) as sumatotalvotos
                        FROM    VOTACION vot1, ELECCION ele1, LUGAR lu1, LUGAR lu2, LUGAR lu3,
LUGAR lu4,
                                        TIPOLUGAR tp1, TIPOLUGAR tp2, TIPOLUGAR tp3, TIPOLUGAR
tp4, POLITICO pol1
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                        lu1.idtipolugar = tp1.idtipolugar AND
                                        tp2.nombtipolugar like '%Region%' AND
                                        lu2.idtipolugar = tp2.idtipolugar AND
                                        tp3.nombtipolugar like '%Departamento%' AND
                                        lu3.idtipolugar = tp3.idtipolugar AND
                                        tp4.nombtipolugar like '%Municipio%' AND
                                        lu4.idtipolugar = tp4.idtipolugar AND
                                        lu1.idlugar1 = lu2.idlugar2 AND
                                        lu2.idlugar1 = lu3.idlugar2 AND
                                        lu3.idlugar1 = lu4.idlugar2 AND
                                        vot1.ideleccion = ele1.ideleccion AND
                                        ele1.idlugar1 = lu4.idlugar1 AND
                                        vot1.idpolitico = pol1.idpolitico
                        GROUP BY lu1.idlugar1, lu4.idlugar1, pol1.idpolitico
                        ORDER BY lu1.idlugar1, lu4.idlugar1, pol1.idpolitico, sumatotalvotos DESC
                )       TABTO
                GROUP BY TABTO.idpais, TABTO.idmuni
        )       TABTI,
        (
                SELECT  lu1.idlugar1 as idpais, lu4.idlugar1 as idmuni, pol1.idpolitico as idpol,
                                SUM(vot1.cantvoto) as sumatotalvotos
                FROM    VOTACION vot1, ELECCION ele1, LUGAR lu1, LUGAR lu2, LUGAR lu3, LUGAR lu4,
                        TIPOLUGAR tp1, TIPOLUGAR tp2, TIPOLUGAR tp3, TIPOLUGAR tp4, POLITICO pol1
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2 AND
                                vot1.ideleccion = ele1.ideleccion AND
                                ele1.idlugar1 = lu4.idlugar1 AND
                                vot1.idpolitico = pol1.idpolitico
                GROUP BY lu1.idlugar1, lu4.idlugar1, pol1.idpolitico
                ORDER BY lu1.idlugar1, lu4.idlugar1, pol1.idpolitico, sumatotalvotos DESC
        )       TABTM, LUGAR lug1, POLITICO pol1
        WHERE  TABTI.idpais = TABTM.idpais and
                        TABTI.idmuni = TABTM.idmuni and
                        TABTI.maxvotos = TABTM.sumatotalvotos and
                        TABTI.idpais = lug1.idlugar1 and
                        TABTM.idpol = pol1.idpolitico
        GROUP BY lug1.nomblugar, pol1.nombpolitico
        ORDER BY numalc DESC, lug1.nomblugar, pol1.nombpolitico
) BAM,
(
        SELECT BOM.nomblugarp as nomblugarp, MAX(BOM.numalc) as maxalc
        FROM
        (
                SELECT lug1.nomblugar as nomblugarp, pol1.nombpolitico as nombpoliticop,
                                COUNT(TABTI.idmuni) as numalc
                FROM
```


```
                (
                        SELECT TABTO.idpais as idpais, TABTO.idmuni as idmuni, MAX(TABTO.sumatotalvotos)
                                        as maxvotos
                        FROM
                        (
                                SELECT  lu1.idlugar1 as idpais, lu4.idlugar1 as idmuni, pol1.idpolitico as idpol,
                                                SUM(vot1.cantvoto) as sumatotalvotos
                                FROM    VOTACION vot1, ELECCION ele1, LUGAR lu1, LUGAR lu2, LUGAR
lu3, LUGAR lu4,
                                                TIPOLUGAR tp1, TIPOLUGAR tp2, TIPOLUGAR tp3,
TIPOLUGAR tp4, POLITICO pol1
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2 AND
                                                vot1.ideleccion = ele1.ideleccion AND
                                                ele1.idlugar1 = lu4.idlugar1 AND
                                                vot1.idpolitico = pol1.idpolitico
                                GROUP BY lu1.idlugar1, lu4.idlugar1, pol1.idpolitico
                                ORDER BY lu1.idlugar1, lu4.idlugar1, pol1.idpolitico, sumatotalvotos DESC
                        )       TABTO
                        GROUP BY TABTO.idpais, TABTO.idmuni
                )       TABTI,
                (
                        SELECT  lu1.idlugar1 as idpais, lu4.idlugar1 as idmuni, pol1.idpolitico as idpol,
                                        SUM(vot1.cantvoto) as sumatotalvotos
                        FROM    VOTACION vot1, ELECCION ele1, LUGAR lu1, LUGAR lu2, LUGAR lu3,
LUGAR lu4,
                                        TIPOLUGAR tp1, TIPOLUGAR tp2, TIPOLUGAR tp3, TIPOLUGAR
tp4, POLITICO pol1
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                        lu1.idtipolugar = tp1.idtipolugar AND
                                        tp2.nombtipolugar like '%Region%' AND
                                        lu2.idtipolugar = tp2.idtipolugar AND
                                        tp3.nombtipolugar like '%Departamento%' AND
                                        lu3.idtipolugar = tp3.idtipolugar AND
                                        tp4.nombtipolugar like '%Municipio%' AND
                                        lu4.idtipolugar = tp4.idtipolugar AND
                                        lu1.idlugar1 = lu2.idlugar2 AND
                                        lu2.idlugar1 = lu3.idlugar2 AND
                                        lu3.idlugar1 = lu4.idlugar2 AND
                                        vot1.ideleccion = ele1.ideleccion AND
                                        ele1.idlugar1 = lu4.idlugar1 AND
                                        vot1.idpolitico = pol1.idpolitico
                        GROUP BY lu1.idlugar1, lu4.idlugar1, pol1.idpolitico
                        ORDER BY lu1.idlugar1, lu4.idlugar1, pol1.idpolitico, sumatotalvotos DESC
                )       TABTM, LUGAR lug1, POLITICO pol1
                WHERE  TABTI.idpais = TABTM.idpais and
                                TABTI.idmuni = TABTM.idmuni and
                                TABTI.maxvotos = TABTM.sumatotalvotos and
                                TABTI.idpais = lug1.idlugar1 and
                                TABTM.idpol = pol1.idpolitico
                GROUP BY lug1.nomblugar, pol1.nombpolitico
                ORDER BY numalc DESC, lug1.nomblugar, pol1.nombpolitico
        ) BOM
        GROUP BY BOM.nomblugarp
) BIM
WHERE  BAM.nomblugarp = BIM.nomblugarp AND
```


```
        BAM.numalc = BIM.maxalc
ORDER BY BAM.numalc DESC
;
```

	NOMBRE PAIS	NOMBRE PARTIDO	# ALCALDIAS GANADAS
1	GUATEMALA	Partido Union Por Guat.	122
2	HONDURAS	Partido Liberal de Hon.	105
3	EL SALVAD...	Farabundo Martí lib. Nac.	92
4	Nicaragua	Partido Socialista Nic.	51
5	Panama	Panama Libre	33
6	Costa Rica	Partido Civil	28

3.4.2.6. Consulta No. 6

Desplegar todas las regiones por pais en las que predomina la raza indigena. Es decir, hay más votos que las otras razas.

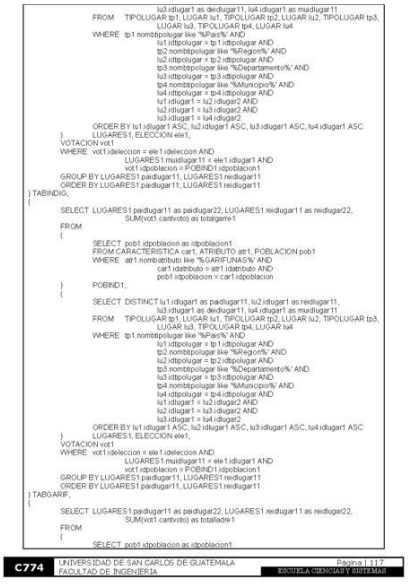
La consulta principal se encuentra integrada por tres subconsultas. La primer subconsulta devuelve el pais, la región y el total de votos obtenidos por la población indigena. La segunda subconsulta permite devolver el pais, la región y el total de votos obtenidos por la población garifuna. La tercer subconsulta devuelve el pais, la región y el total de votos obtenidos por la población ladina. Las tres subconsultas cada una se encuentra integrada por dos subconsultas. La primera devuelve el tipo de población y la segunda los paises, regiones, departamentos y municipios.

La consulta principal a partir de estas tres subconsultas puede determinar por medio de comparaciones cual población ha obtenido más votos en una región de un pais. La figura 3.48. muestra esta consulta.

Figura 3.48. Script consulta 6

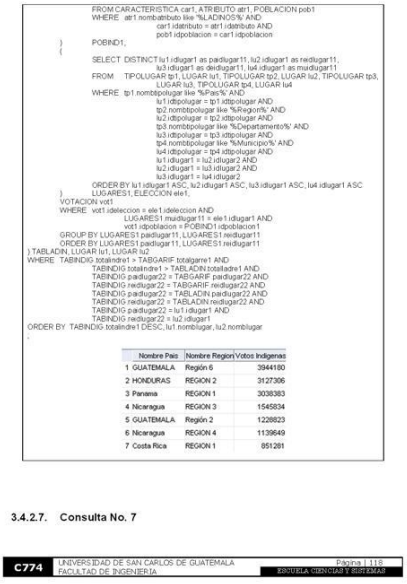
```
SELECT  DISTINCT lu1.nomblugar as "Nombre Pais", lu2.nomblugar as "Nombre Region",
                TABINDIG.totalindre1 as "Votos Indigenas"
FROM
(
        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.reidlugar11 as reidlugar22,
                        SUM(vot1.cantvoto) as totalindre1
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%INDIGENAS%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )       POBIND1,
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
```

Figura 107. Páginas 117 – 120 Libro “Bases de datos principiantes”



```
                                    lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                    LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                    lu1.idtipolugar = tp1.idtipolugar AND
                                    tp2.nombtipolugar like '%Region%' AND
                                    lu2.idtipolugar = tp2.idtipolugar AND
                                    tp3.nombtipolugar like '%Departamento%' AND
                                    lu3.idtipolugar = tp3.idtipolugar AND
                                    tp4.nombtipolugar like '%Municipio%' AND
                                    lu4.idtipolugar = tp4.idtipolugar AND
                                    lu1.idlugar1 = lu2.idlugar2 AND
                                    lu2.idlugar1 = lu3.idlugar2 AND
                                    lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                    LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                    vot1.idpoblacion = POBIND1.idpoblacion1
        GROUP BY LUGARES1.padlugar11, LUGARES1.reidlugar11
        ORDER BY LUGARES1.padlugar11, LUGARES1.reidlugar11
) TABINDIG,
(
        SELECT  LUGARES1.padlugar11 as padlugar22, LUGARES1.reidlugar11 as reidlugar22,
                                    SUM(vot1.cantvoto) as totalgarre1

        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%GARIFUNAS%' AND
                                    car1.idatributo = atr1.idatributo AND
                                    pob1.idpoblacion = car1.idpoblacion
        )       POBIND1,
        (
                SELECT  DISTINCT lu1.idlugar1 as padlugar11, lu2.idlugar1 as reidlugar11,
                                    lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                    LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                    lu1.idtipolugar = tp1.idtipolugar AND
                                    tp2.nombtipolugar like '%Region%' AND
                                    lu2.idtipolugar = tp2.idtipolugar AND
                                    tp3.nombtipolugar like '%Departamento%' AND
                                    lu3.idtipolugar = tp3.idtipolugar AND
                                    tp4.nombtipolugar like '%Municipio%' AND
                                    lu4.idtipolugar = tp4.idtipolugar AND
                                    lu1.idlugar1 = lu2.idlugar2 AND
                                    lu2.idlugar1 = lu3.idlugar2 AND
                                    lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                    LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                    vot1.idpoblacion = POBIND1.idpoblacion1
        GROUP BY LUGARES1.padlugar11, LUGARES1.reidlugar11
        ORDER BY LUGARES1.padlugar11, LUGARES1.reidlugar11
) TABGARIF,
(
        SELECT  LUGARES1.padlugar11 as padlugar22, LUGARES1.reidlugar11 as reidlugar22,
                                    SUM(vot1.cantvoto) as totalladre1
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
```

```
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%LADINOS%' AND
                                    car1.idatributo = atr1.idatributo AND
                                    pob1.idpoblacion = car1.idpoblacion
        )       POBIND1,
        (
                SELECT  DISTINCT lu1.idlugar1 as padlugar11, lu2.idlugar1 as reidlugar11,
                                    lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                    LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                    lu1.idtipolugar = tp1.idtipolugar AND
                                    lu2.idtipolugar = tp2.idtipolugar AND
                                    tp3.nombtipolugar like '%Departamento%' AND
                                    lu3.idtipolugar = tp3.idtipolugar AND
                                    tp4.nombtipolugar like '%Municipio%' AND
                                    lu4.idtipolugar = tp4.idtipolugar AND
                                    lu1.idlugar1 = lu2.idlugar2 AND
                                    lu2.idlugar1 = lu3.idlugar2 AND
                                    lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                    LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                    vot1.idpoblacion = POBIND1.idpoblacion1
        GROUP BY LUGARES1.padlugar11, LUGARES1.reidlugar11
        ORDER BY LUGARES1.padlugar11, LUGARES1.reidlugar11
) TABLADIN, LUGAR lu1, LUGAR lu2
WHERE  TABINDIG.totalindre1 > TABGARIF.totalgarre1 AND
                    TABINDIG.totalindre1 > TABLADIN.totalladre1 AND
                    TABINDIG.padlugar22 = TABGARIF.padlugar22 AND
                    TABINDIG.reidlugar22 = TABGARIF.reidlugar22 AND
                    TABINDIG.padlugar22 = TABLADIN.padlugar22 AND
                    TABINDIG.reidlugar22 = TABLADIN.reidlugar22 AND
                    TABINDIG.padlugar22 = lu1.idlugar1 AND
                    TABINDIG.reidlugar22 = lu2.idlugar1
ORDER BY  TABINDIG.totalindre1 DESC, lu1.nomblugar, lu2.nomblugar
;
```

	Nombre Pais	Nombre Region	Votos Indigenas
1	GUATEMALA	Región 6	3944180
2	HONDURAS	REGION 2	3127306
3	Panama	REGION 1	3038383
4	Nicaragua	REGION 3	1545834
5	GUATEMALA	Región 2	1228923
6	Nicaragua	REGION 4	1139649
7	Costa Rica	REGION 1	851281

3.4.2.7. Consulta No. 7

Desplegar el porcentaje de mujeres universitarias y hombres universitarios que votaron por departamento, donde las mujeres universitarias que votaron fueron más que los hombres universitarios que votaron.

La consulta se toma a partir únicamente del porcentaje total de las poblaciones universitarias que votaron. Esto quiere decir que tanto las mujeres como los hombres universitarios sumarán 100% en sus porcentajes de votación.

La consulta principal se encuentra integrada por dos subconsultas. La primer subconsulta devuelve el país, el departamento y el total de votos obtenidos por las mujeres universitarias. La segunda subconsulta devuelve el país, el departamento y el total de votos obtenidos por los hombres universitarios. Las subconsultas se encuentran integradas cada una por dos subconsultas. La primera devuelve las poblaciones que son universitarias y hombres o mujeres según sea el caso. La segunda devuelve el país, región, departamento y municipio.

La consulta principal realiza comparaciones y despliega todos los casos donde las mujeres que votaron fueron más que los hombres. Estos tomando en cuenta el departamento, el género y que son poblaciones universitarias. La figura 3.49. muestra esta consulta.

Figura 3.49. Script consulta 6

```
SELECT lugp.nomblugar as "NOMBRE PAIS", lugd.nomblugar as "NOMBRE DEPTO",
                round((TABMUJ.totalunivm1*100)/(TABMUJ.totalunivm1+TABHOM.totalunivh1),2)
                        as "% MUJ VOTOS", 100-
round((TABMUJ.totalunivm1*100)/(TABMUJ.totalunivm1+TABHOM.totalunivh1),2)
                        as "% HOM VOTOS"
FROM
(
        SELECT  LUGARES1.padlugar11 as padlugar22, LUGARES1.deidlugar11 as deidlugar22,
                                SUM(vot1.cantvoto) as totalunivm1
        FROM
        (
                SELECT PALF5.idpoblacion5 as idpoblacion1
                FROM
                (
                        SELECT  DISTINCT pob5.idpoblacion as idpoblacion5,
                        FROM CARACTERISTICA car5, ATRIBUTO atr5, POBLACION pob5,
                        (
                                SELECT  rta5.idatributo as rta1idatributo
                                FROM    ATRIBUTO rta5
                                WHERE  rta5.nombatributo like '%UNIVERSITARIOS%'
                        )       ALFTAB5
                        WHERE  atr5.idatributo = ALFTAB5.rta1idatributo AND
                                                car5.idatributo = atr5.idatributo AND
                                                pob5.idpoblacion = car5.idpoblacion
                        ORDER BY pob5.idpoblacion
                ) PALF5,
                (
                        SELECT  pob5.idpoblacion as idpoblacion5
                        FROM CARACTERISTICA car5, ATRIBUTO atr5, POBLACION pob5
```


```
                        WHERE  atr5.nombatributo like '%mujeres%' AND
                                                car5.idatributo = atr5.idatributo AND
                                                pob5.idpoblacion = car5.idpoblacion
                ) MUJE5
                WHERE PALF5.idpoblacion5 = MUJE5.idpoblacion5
                ORDER BY PALF5.idpoblacion5 ASC
        )       POBUNI1,
        (
                SELECT  DISTINCT lu1.idlugar1 as padlugar11, lu2.idlugar1 as reidlugar11,
                                    lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                    LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                    lu1.idtipolugar = tp1.idtipolugar AND
                                    tp2.nombtipolugar like '%Region%' AND
                                    lu2.idtipolugar = tp2.idtipolugar AND
                                    tp3.nombtipolugar like '%Departamento%' AND
                                    lu3.idtipolugar = tp3.idtipolugar AND
                                    tp4.nombtipolugar like '%Municipio%' AND
                                    lu4.idtipolugar = tp4.idtipolugar AND
                                    lu1.idlugar1 = lu2.idlugar2 AND
                                    lu2.idlugar1 = lu3.idlugar2 AND
                                    lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                    LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                    vot1.idpoblacion = POBUNI1.idpoblacion1
        GROUP BY LUGARES1.padlugar11, LUGARES1.deidlugar11
        ORDER BY LUGARES1.padlugar11, LUGARES1.deidlugar11
) TABMUJ,
(
        SELECT  LUGARES1.padlugar11 as padlugar22, LUGARES1.deidlugar11 as deidlugar22,
                                SUM(vot1.cantvoto) as totalunivh1
        FROM
        (
                SELECT PALF5.idpoblacion5 as idpoblacion1
                FROM
                (
                        SELECT  DISTINCT pob5.idpoblacion as idpoblacion5
                        FROM CARACTERISTICA car5, ATRIBUTO atr5, POBLACION pob5,
                        (
                                SELECT  rta5.idatributo as rta1idatributo
                                FROM    ATRIBUTO rta5
                                WHERE  rta5.nombatributo like '%UNIVERSITARIOS%'
                        )       ALFTAB5
                        WHERE  atr5.idatributo = ALFTAB5.rta1idatributo AND
                                                car5.idatributo = atr5.idatributo AND
                                                pob5.idpoblacion = car5.idpoblacion
                        ORDER BY pob5.idpoblacion
                ) PALF5,
                (
                        SELECT  pob5.idpoblacion as idpoblacion5
                        FROM CARACTERISTICA car5, ATRIBUTO atr5, POBLACION pob5
                        WHERE  atr5.nombatributo like '%hombres%' AND
                                                car5.idatributo = atr5.idatributo AND
                                                pob5.idpoblacion = car5.idpoblacion
                ) HOMB5
                WHERE PALF5.idpoblacion5 = HOMB5.idpoblacion5
                ORDER BY PALF5.idpoblacion5 ASC
        )       POBUNI1,
        (
                SELECT  DISTINCT lu1.idlugar1 as padlugar11, lu2.idlugar1 as reidlugar11,
                                    lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
```

Figura 108. Páginas 121 – 124 Libro “Bases de datos principiantes”


```
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                        lu1.idtipolugar = tp1.idtipolugar AND
                        tp2.nombtipolugar like '%Region%' AND
                        lu2.idtipolugar = tp2.idtipolugar AND
                        tp3.nombtipolugar like '%Departamento%' AND
                        lu3.idtipolugar = tp3.idtipolugar AND
                        tp4.nombtipolugar like '%Municipio%' AND
                        lu4.idtipolugar = tp4.idtipolugar AND
                        lu1.idlugar1 = lu2.idlugar2 AND
                        lu2.idlugar1 = lu3.idlugar2 AND
                        lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE   vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBUNI1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11, LUGARES1.deidlugar11
        ORDER BY LUGARES1.paidlugar11, LUGARES1.deidlugar11
) TABHOM, LUGAR lugd, LUGAR lugp
WHERE   TABMUJ.paidlugar22 = TABHOM.paidlugar22 AND
                TABMUJ.deidlugar22 = TABHOM.deidlugar22 AND
                TABMUJ.totalvinm1 = TABHOM.totalvinh1 AND
                TABMUJ.deidlugar22 = lugd.idlugar1 AND
        TABMUJ.paidlugar22 = lugp.idlugar1
ORDER BY lugp.nomblugar asc, lugd.nomblugar asc, "% MUJ VOTOS" desc
;
```

	NOMBRE PAIS	NOMBRE DEPTO	% MUJ VOTOS	% HOM VOTOS
1	Costa Rica	Alajuela	52,48	47,52
2	Costa Rica	Cartago	51,29	48,71
3	Costa Rica	Guanacaste	51,13	48,87
4	EL SALVADOR	Cabañas	52,85	47,15
5	EL SALVADOR	Chalatenango	51,31	48,69
6	EL SALVADOR	La Paz	50,77	49,23
7	EL SALVADOR	La Unión	50,67	49,33
8	EL SALVADOR	San Vicente	51,22	48,78
9	EL SALVADOR	Sonsonate	50,09	49,91
10	EL SALVADOR	Usulutlán	51,16	48,84

3.4.2.8. Consulta No. 8

Desplegar la lista de departamentos de Guatemala y número de votos obtenidos, para los departamentos que obtuvieron más votos que el departamento de Guatemala.

La consulta principal se encuentra integrada por dos subconsultas. La primera devuelve la cantidad de votos del departamento de Guatemala. La

segunda consulta devuelve la cantidad de votos obtenidos en todos los departamentos de Guatemala. Ambas subconsultas se encuentran integradas por otra que permite el cálculo de votos del pais Guatemala para los departamentos o para el departamento de Guatemala.

La consulta principal entonces será la encargada de realizar la comparación y listar los departamentos cuyos votos son mayores al departamento de Guatemala. La figura 3.50. muestra esta consulta.

Figura 3.50. Script consulta 8

```
SELECT lugs1.nomblugar as "DEPTO VOTOS > DEPTO GUATE", TOTALVOTDG.totaldeptog
                as "VOTOS OBTENIDOS EN DEPTO"
FROM
(
        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.deidlugar11 as deidlugar22,
                        SUM(vot1.cantvoto) as totaldeptog
        FROM
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2 AND
                                lu1.nomblugar like '%GUATEMALA%'
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1,
                ELECCION ele1,
                POBLACION       pob1,
                VOTACION vot1,
        WHERE   vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = pob1.idpoblacion
        GROUP BY LUGARES1.paidlugar11, LUGARES1.deidlugar11
        ORDER BY LUGARES1.paidlugar11, LUGARES1.deidlugar11
        TOTALVOTDG;
(
        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.deidlugar11 as deidlugar22,
                        SUM(vot1.cantvoto) as totaldeptog
        FROM
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
```



```
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2 AND
                                lu1.nomblugar like '%GUATEMALA%'
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )       LUGARES1,
                        ELECCION ele1,
                        POBLACION       pob1,
                        VOTACION vot1,
                        LUGAR lu1
                WHERE   vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = pob1.idpoblacion AND
                                LUGARES1.deidlugar11 = lu1.idlugar1 AND
                                lu1.nomblugar like '%Guatemala%'
                GROUP BY LUGARES1.paidlugar11, LUGARES1.deidlugar11
                ORDER BY LUGARES1.paidlugar11, LUGARES1.deidlugar11
        )       TOTALVOTDGG, LUGAR lugs1
        WHERE TOTALVOTDG.totaldeptog > TOTALVOTDGG.totaldeptog AND
                TOTALVOTDG.deidlugar22 = lugs1.idlugar1
ORDER BY TOTALVOTDG.totaldeptog DESC, lugs1.nomblugar ASC
;
```

	DEPTO VOTOS > DEPTO GUATE	VOTOS OBTENIDOS EN DEPTO
1	Huehuetenango	3359427
2	San Marcos	3110806
3	Quetzaltenango	2593013
4	Quiche	2264208
5	Suchitepequez	2114918
6	Solola	2061749
7	Jutiapa	1833716

3.4.2.9. Consulta No. 9

Desplegar el nombre del pais, la región y el promedio de votos por departamento. Por ejemplo: Si la región tiene tres departamentos, se debe sumar todos los votos de la región y dividirlo dentro de tres (número de departamentos de la región).

La consulta principal se encuentra integrada por una subconsulta. Esta subconsulta se encarga de devolver el pais, la región, el departamento y la cantidad de votos obtenidos en ese departamento. La consulta principal entonces se encarga de sacar el promedio de los votos obtenidos por departamento, pero se agrupa todo por pais y por región lo que permite obtener el resultado de la consulta. La figura 3.51. muestra esta consulta.

Figura 3.51. Script consulta 9


```
SELECT lu1.nomblugar AS "NOMBRE PAIS", lu2.nomblugar AS "NOMBRE REGION",
                ROUND(AVG(TABDEPV.totaldeptog),2) AS "PROM X REG EN DEPTO"
FROM
(
        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.reidlugar11 as reidlugar22,
                        LUGARES1.deidlugar11 as deidlugar22, SUM(vot1.cantvoto) as totaldeptog
        FROM
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1,
                ELECCION ele1,
                POBLACION       pob1,
                VOTACION vot1,
                LUGAR lu1
        WHERE   vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = pob1.idpoblacion AND
                        LUGARES1.deidlugar11 = lu1.idlugar1
        GROUP BY LUGARES1.paidlugar11, LUGARES1.deidlugar11, LUGARES1.reidlugar11
        ORDER BY LUGARES1.paidlugar11, LUGARES1.deidlugar11, LUGARES1.reidlugar11
) TABDEPV, LUGAR lu1, LUGAR lu2
WHERE   TABDEPV.paidlugar22 = lu1.idlugar1 AND
        TABDEPV.reidlugar22 = lu2.idlugar1
GROUP BY lu1.nomblugar, lu2.nomblugar
ORDER BY lu1.nomblugar, lu2.nomblugar
;
```

	NOMBRE PAIS	NOMBRE DEPTO	PROM X REG EN DEPTO
1	Costa Rica	REGION 1	1242587
2	Costa Rica	REGION 2	1147714
3	Costa Rica	REGION 3	924873,5
4	EL SALVAD...	REGION 1	967996
5	EL SALVAD...	REGION 2	2132939,2
6	EL SALVAD...	REGION 3	2114541
7	EL SALVAD...	REGION 4	2337229,67
8	GUATEMALA	Región 1	1796343
9	GUATEMALA	Región 2	911048,5
10	GUATEMALA	Región 4	1376273,33

Figura 109. Páginas 125 – 128 Libro “Bases de datos principiantes”

3.4.2.10. Consulta No. 10

Desplegar el total de votos de los municipios agrupados por su letra inicial. Es decir, agrupar todos los municipios con letra A y calcular su número de votos, lo mismo para los de letra inicial B, y así sucesivamente hasta la Z.

La consulta principal se encuentra integrada por dos subconsultas. La primer subconsulta despliega la primera letra de todos los municipios. La segunda subconsulta devuelve el total de votos obtenidos por municipio y por país. La consulta principal utiliza la función *substr* para extraer la primer letra de los municipios y de esta forma calcular el total de votos por letra de municipio. La figura 3.52. muestra esta consulta.

Figura 3.52. Script consulta 10

```
SELECT TABSUBSTR.primletter as "LETRA INICIAL MUNIC", SUM(TABMU.totalmunic)
            as "TOTAL VOTOS X LETRA"
FROM
(
        SELECT DISTINCT SUBSTR(lug1.nomblugar, 1, 1) as primletter
        FROM LUGAR lug1
        WHERE  lug1.idtipolugar = 4
        ORDER BY primletter
) TABSUBSTR,
(
        SELECT  TABMUN.paidlugar22 as paidlugar33, luga1.nomblugar as nombmuni,
                TABMUN.totalmunig as totalmunic
        FROM
        (
                SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
                        muidlugar22, SUM(vot1.cantvoto) as totalmunig
                FROM
                (
                        SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as dedlugar11, lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                ) LUGARES1,
                ELECCION ele1,
                POBLACION pob1,
                VOTACION vot1
                WHERE  vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = pob1.idpoblacion
                GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11
```



```
                ORDER BY LUGARES1.paidlugar11, LUGARES1.muidlugar11
        ) TABMUN, LUGAR luga1
        WHERE  TABMUN.muidlugar22 = luga1.idlugar1
)       TABMU
WHERE   TABSUBSTR.primletter = SUBSTR(TABMU.nombmuni, 1, 1)
GROUP BY TABSUBSTR.primletter
ORDER BY "LETRA INICIAL MUNIC" ASC
;
```

LETRA INICIAL MUNIC	TOTAL VOTOS X LETRA
1 A	5059662
2 B	2656501
3 C	15118064
4 D	2373926
5 E	6693631
6 F	650675
7 G	3513640
8 H	755272
9 I	1224959

3.4.2.11. Consulta No. 11

Desplegar el nombre del municipio y el nombre de los dos partidos políticos con más votos en el municipio, ordenados por país.

La consulta principal se compone de otra subconsulta. Esta subconsulta posee dos subconsultas. La primera obtiene los partidos por municipio y sus votos. Esta subconsulta utiliza la operación *minus* para poder sacar los máximos y la otra subconsulta obtiene todos los mínimos. La segunda subconsulta muestra todos los máximos e incluye su total de votos. Esto permite que la consulta principal pueda desplegar los dos partidos políticos con más votos obtenidos en el municipio. Esto obviara todos los demás partidos. La figura 3.53. muestra esta consulta.

Figura 3.53. Script consulta 11

```
SELECT  lug9.nomblugar as "NOMBRE PAIS", lug10.nomblugar as "NOMBRE MUNICIPIO",
                YUU.nombpolitico44 as "NOMBRE PARTIDO", YUU.numvot as "# VOTOS"
FROM
(
        SELECT LUGARES1.paidlugar11 as paidlugar44, LUGARES1.muidlugar11 as muidlugar44,
                par1.nombpolitico as nombpolitico44, SUM(vot1.cantvoto) as numvot
        FROM
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                        lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
```

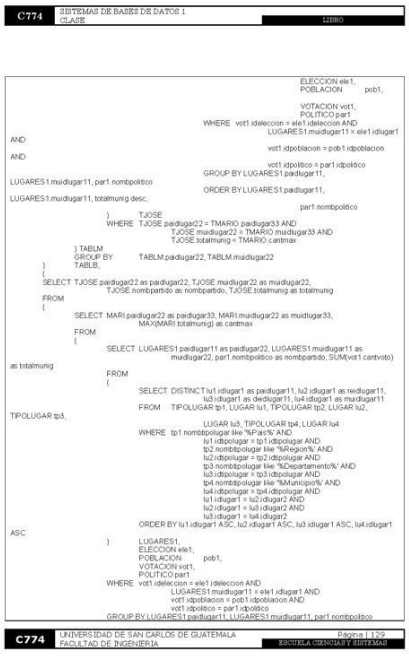


```
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                        lu1.idtipolugar = tp1.idtipolugar AND
                        tp2.nombtipolugar like '%Region%' AND
                        lu2.idtipolugar = tp2.idtipolugar AND
                        tp3.nombtipolugar like '%Departamento%' AND
                        lu3.idtipolugar = tp3.idtipolugar AND
                        tp4.nombtipolugar like '%Municipio%' AND
                        lu4.idtipolugar = tp4.idtipolugar AND
                        lu1.idlugar1 = lu2.idlugar2 AND
                        lu2.idlugar1 = lu3.idlugar2 AND
                        lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        ) LUGARES1,
        ELECCION ele1,
        POBLACION pob1,
        VOTACION vot1,
        POLITICO par1
WHERE  vot1.ideleccion = ele1.ideleccion AND
        LUGARES1.muidlugar11 = ele1.idlugar1 AND
        vot1.idpoblacion = pob1.idpoblacion AND
        vot1.idpolitico = par1.idpolitico
GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11, par1.nombpolitico
MINUS
SELECT  TABLJ.paidlugar22 as paidlugar22, TABLJ.muidlugar22 as muidlugar22,
                TABLJ.nombpartido as nombpartido, TABLJ.totalmunig as totalmunig
FROM
(
        SELECT  TABLM.paidlugar22 as paidlugar22, TABLM.muidlugar22 as muidlugar22,
                        MAX(TABLM.totalmunig) as cantmax
        FROM
        (
                SELECT  TJOSE.paidlugar22 as paidlugar22, TJOSE.muidlugar22 as muidlugar22,
                                TJOSE.nombpartido as nombpartido, TJOSE.totalmunig as totalmunig
                FROM
                (
                        SELECT  MARI.paidlugar22 as paidlugar33, MARI.muidlugar22 as muidlugar33,
                                        MAX(MARI.totalmunig) as cantmax
                        FROM
                        (
                                SELECT  LUGARES1.paidlugar11 as paidlugar22,
LUGARES1.muidlugar11 as
                                        muidlugar22, par1.nombpolitico as nombpartido,
SUM(vot1.cantvoto)
                                        as totalmunig
                                FROM
                                (
                                        SELECT  DISTINCT lu1.idlugar1 as paidlugar11,
lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as dedlugar11, lu4.idlugar1
as muidlugar11
                                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2,
LUGAR lu2, TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR
lu4
                                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like
'%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%'
AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
```



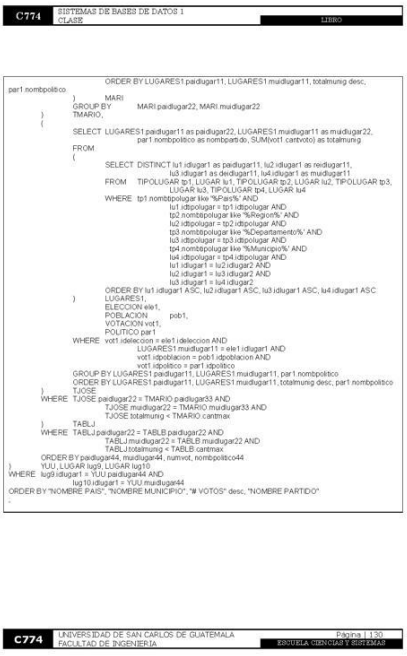
```
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1
ASC, lu4.idlugar1 ASC
                                ) LUGARES1,
                                ELECCION ele1,
                                POBLACION pob1,
                                VOTACION vot1,
                                POLITICO par1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = pob1.idpoblacion AND
                                vot1.idpolitico = par1.idpolitico
                        GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11,
par1.nombpolitico
                        ORDER BY LUGARES1.paidlugar11, LUGARES1.muidlugar11,
totalmunig desc, par1.nombpolitico
                ) MARI
                GROUP BY MARI.paidlugar22, MARI.muidlugar22
        ) TMARIO,
        (
                SELECT  LUGARES1.paidlugar11 as paidlugar22,
                        muidlugar22, par1.nombpolitco as
nombpartido, SUM(vot1.cantvoto)
                        as totalmunig
                FROM
                (
                        SELECT  DISTINCT
lu1.idlugar1 as paidlugar11, lu2.idlugar1
                                as
reidlugar11, lu3.idlugar1 as dedlugar11,
                                lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1,
LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
                                TIPOLUGAR tp3, LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE  tp1.nombtipolugar
like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC,
lu2.idlugar1 ASC, lu3.idlugar1 ASC,
                                lu4.idlugar1 ASC
                ) LUGARES1,
```

Figura 110. Páginas 129 – 132 Libro “Bases de datos principiantes”

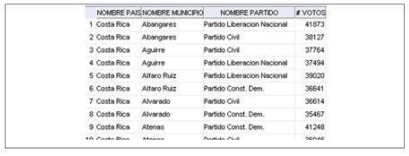


```
                                                                ELECCION ele1,
                                                                POBLACION       pob1,

                                                                VOTACION vot1,
                                                                POLITICO par1
                                                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                                                LUGARES1.muidlugar11 = ele1.idlugar1
AND
                                                                vot1.idpoblacion = pob1.idpoblacion
AND
                                                                vot1.idpolitico = par1.idpolitico
                                                        GROUP BY LUGARES1.paidlugar11,
LUGARES1.muidlugar11, par1.nombpolitico
                                                        ORDER BY LUGARES1.paidlugar11,
LUGARES1.muidlugar11, totalmunig desc,
                                                                par1.nombpolitico
                        )       TJOSE
                        WHERE  TJOSE.paidlugar22 = TMARIO.paidlugar33 AND
                                TJOSE.muidlugar22 = TMARIO.muidlugar33 AND
                                TJOSE.totalmunig < TMARIO.cantmax
                ) TABLM
                GROUP BY        TABLM.paidlugar22, TABLM.muidlugar22
        )
        TABLB;
        {
        SELECT TJOSE.paidlugar22 as paidlugar22, TJOSE.muidlugar22 as muidlugar22,
                TJOSE.nombpartido as nombpartido, TJOSE.totalmunig as totalmunig
        FROM
        (
                SELECT  MARI.paidlugar22 as paidlugar33, MARI.muidlugar22 as muidlugar33,
                        MAX(MARI.totalmunig) as cantmax
                FROM
                (
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
                                muidlugar22, par1.nombpolitico as nombpartido, SUM(vot1.cantvoto)
as totalmunig
                        FROM
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                        lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                        lu1.idtipolugar = tp1.idtipolugar AND
                                        tp2.nombtipolugar like '%Region%' AND
                                        lu2.idtipolugar = tp2.idtipolugar AND
                                        tp3.nombtipolugar like '%Departamento%' AND
                                        lu3.idtipolugar = tp3.idtipolugar AND
                                        tp4.nombtipolugar like '%Municipio%' AND
                                        lu4.idtipolugar = tp4.idtipolugar AND
                                        lu1.idlugar1 = lu2.idlugar2 AND
                                        lu2.idlugar1 = lu3.idlugar2 AND
                                        lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1,
                                ELECCION ele1,
                                POBLACION       pob1,
                                VOTACION vot1,
                                POLITICO par1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = pob1.idpoblacion AND
                                vot1.idpolitico = par1.idpolitico
                        GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11, par1.nombpolitico
```



```
                        ORDER BY LUGARES1.paidlugar11, LUGARES1.muidlugar11, totalmunig desc,
par1.nombpolitico
                )       MARI
                GROUP BY        MARI.paidlugar22, MARI.muidlugar22
        )       TMARIO,
        (
                SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as muidlugar22,
                        par1.nombpolitico as nombpartido, SUM(vot1.cantvoto) as totalmunig
                FROM
                (
                        SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )       LUGARES1,
                        ELECCION ele1,
                        POBLACION       pob1,
                        VOTACION vot1,
                        POLITICO par1
                WHERE  vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = pob1.idpoblacion AND
                        vot1.idpolitico = par1.idpolitico
                GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11, par1.nombpolitico
                ORDER BY LUGARES1.paidlugar11, LUGARES1.muidlugar11, totalmunig desc, par1.nombpolitico
        )       TJOSE
        WHERE  TJOSE.paidlugar22 = TMARIO.paidlugar33 AND
                TJOSE.muidlugar22 = TMARIO.muidlugar33 AND
                TJOSE.totalmunig < TMARIO.cantmax
        )       TABLJ
        WHERE  TABLJ.paidlugar22 = TABLB.paidlugar22 AND
                TABLJ.muidlugar22 = TABLB.muidlugar22 AND
                TABLJ.totalmunig < TABLB.cantmax
        ORDER BY paidlugar44, muidlugar44, numvot, nombpolitico44
)       YUU, LUGAR lug9, LUGAR lug10
WHERE  lug9.idlugar1 = YUU.paidlugar44 AND
        lug10.idlugar1 = YUU.muidlugar44
ORDER BY "NOMBRE PAIS", "NOMBRE MUNICIPIO", "# VOTOS" desc, "NOMBRE PARTIDO"
;
```



	NOMBRE PAIS	NOMBRE MUNICIPIO	NOMBRE PARTIDO	# VOTOS
1	Costa Rica	Abangares	Partido Liberacion Nacional	41873
2	Costa Rica	Abangares	Partido Civil	38127
3	Costa Rica	Aguirre	Partido Civil	37764
4	Costa Rica	Aguirre	Partido Liberacion Nacional	37494
5	Costa Rica	Alfaro Ruiz	Partido Liberacion Nacional	39020
6	Costa Rica	Alfaro Ruiz	Partido Const. Dem.	36641
7	Costa Rica	Alvarado	Partido Civil	36614
8	Costa Rica	Alvarado	Partido Const. Dem.	35467
9	Costa Rica	Atenas	Partido Const. Dem.	41248

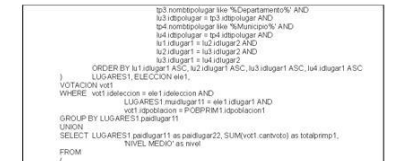
3.4.2.12. Consulta No. 12

Desplegar el total de votos de cada nivel de escolaridad (primario, medio, universitario) por pais, sin importar raza o sexo.

La consulta principal se encuentra integrada por una subconsulta. Esta subconsulta se divide en tres partes. Esto debido a que posee dos unión para poder unir los resultados de tres subconsultas. Estas cada una calcula el total de votantes por pais con escolaridad (primaria, nivel medio, universitario). Estas al unirse despliegan lo solicitado por la consulta principal. La figura 3.54. muestra esta consulta.

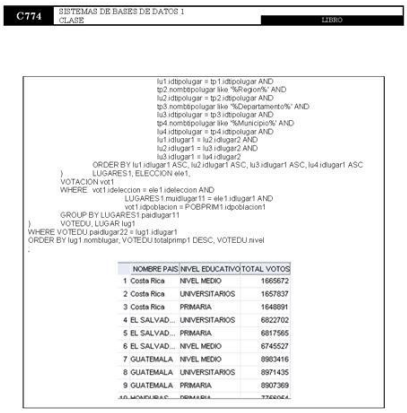
Figura 3.54. Script consulta 12

```
SELECT lug1.nomblugar as "NOMBRE PAIS", VOTEDU.nivel as "NIVEL EDUCATIVO",
                VOTEDU.totalprimp1 as "TOTAL VOTOS"
FROM
(
        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                        'PRIMARIA' as nivel
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%PRIMARIA%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )       POBPRIM1,
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
```



```
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBPRIM1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11
        UNION
        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                        'NIVEL MEDIO' as nivel
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%NIVEL MEDIO%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )       POBPRIM1,
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE  vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBPRIM1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11
        UNION
        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                        'UNIVERSITARIOS' as nivel
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE  atr1.nombatributo like '%UNIVERSITARIOS%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )       POBPRIM1,
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE  tp1.nombtipolugar like '%Pais%' AND
```

Figura 111. Páginas 133 – 136 Libro “Bases de datos principiantes”



```
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE   vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = POBPRIM1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11
)       VOTEDU, LUGAR lug1
WHERE VOTEDU.paidlugar22 = lug1.idlugar1
ORDER BY lug1.nomblugar, VOTEDU.totalprimp1 DESC, VOTEDU.nivel
.
```

	NOMBRE PAIS	NIVEL EDUCATIV	TOTAL VOTOS
1	Costa Rica	NIVEL MEDIO	1665672
2	Costa Rica	UNIVERSITARIOS	1657837
3	Costa Rica	PRIMARIA	1648891
4	EL SALVAD...	UNIVERSITARIOS	6822702
5	EL SALVAD...	PRIMARIA	6817565
6	EL SALVAD...	NIVEL MEDIO	6745527
7	GUATEMALA	NIVEL MEDIO	8983416
8	GUATEMALA	UNIVERSITARIOS	8971435
9	GUATEMALA	PRIMARIA	8907369

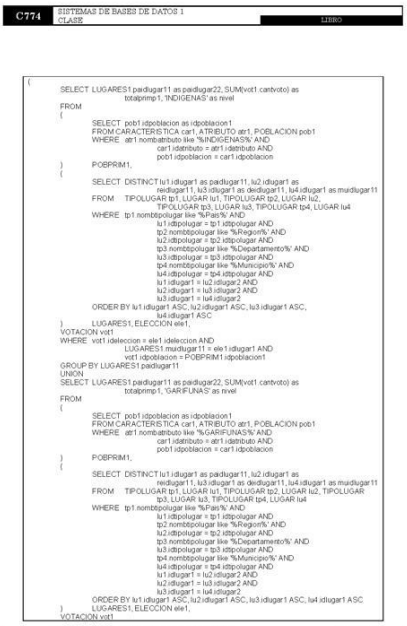
3.4.2.13. Consulta No. 13

Desplegar el nombre del pais y el porcentaje de votos por raza.

La consulta principal se encuentra integrada por una subconsulta. Esta subconsulta se divide en tres partes. Esto debido a que posee dos unión para poder unir los resultados de tres subconsultas. Estas cada una calcula el total de votantes por pais dependiendo su raza (indigena, ladina, garifuna). Estas al unirse despliegan lo solicitado por la consulta principal. La figura 3.55. muestra esta consulta.

Figura 3.55. Script consulta 13

```
SELECT lug1.nomblugar as "NOMBRE PAIS", VOTEDU.nivel as "RAZA",
        round((VOTEDU.totalprimp1*100)/VATO.totalprimp1,2) as "% VOTOS"
FROM
```

```
(
        SELECT LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as
                        totalprimp1, 'INDIGENAS' as nivel
        FROM
        (
                SELECT pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE   atr1.nombatributo like '%INDIGENA%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )       POBPRIM1,
        (
                SELECT DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as
                                reidlugar11, lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
                                TIPOLUGAR tp3, LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC,
                                lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
VOTACION vot1
WHERE   vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBPRIM1.idpoblacion1
GROUP BY LUGARES1.paidlugar11
UNION
SELECT LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as
                totalprimp1, 'GARIFUNAS' as nivel
FROM
(
        SELECT pob1.idpoblacion as idpoblacion1
        FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
        WHERE   atr1.nombatributo like '%GARIFUNAS%' AND
                        car1.idatributo = atr1.idatributo AND
                        pob1.idpoblacion = car1.idpoblacion
)       POBPRIM1,
(
        SELECT DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as
                        reidlugar11, lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR
                        tp3, LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
        WHERE   tp1.nombtipolugar like '%Pais%' AND
                        lu1.idtipolugar = tp1.idtipolugar AND
                        tp2.nombtipolugar like '%Region%' AND
                        lu2.idtipolugar = tp2.idtipolugar AND
                        tp3.nombtipolugar like '%Departamento%' AND
                        lu3.idtipolugar = tp3.idtipolugar AND
                        tp4.nombtipolugar like '%Municipio%' AND
                        lu4.idtipolugar = tp4.idtipolugar AND
                        lu1.idlugar1 = lu2.idlugar2 AND
                        lu2.idlugar1 = lu3.idlugar2 AND
                        lu3.idlugar1 = lu4.idlugar2
        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC,
                        lu4.idlugar1 ASC
)       LUGARES1, ELECCION ele1,
VOTACION vot1
WHERE   vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBPRIM1.idpoblacion1
GROUP BY LUGARES1.paidlugar11
)       VOTEDU,
(
        SELECT LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1
        FROM
        POBLACION       POBPRIM1,
        (
                SELECT DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as
                                reidlugar11, lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR
                                tp3, LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC,
                                lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE   vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = POBPRIM1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11
)       VATO;
LUGAR lug1
WHERE VOTEDU.paidlugar22 = lug1.idlugar1 AND
        VOTEDU.paidlugar22 = VATO.paidlugar22
ORDER BY lug1.nomblugar, VOTEDU.totalprimp1 DESC, VOTEDU.nivel
.
```

	NOMBRE PAIS	RAZA	% VOTOS
1	Costa Rica	INDIGENAS	33,68
2	Costa Rica	GARIFUN...	33,3
3	Costa Rica	LADINOS	33,02
4	EL SALVAD...	GARIFUN...	33,58
5	EL SALVAD...	INDIGENAS	33,41
6	EL SALVAD...	LADINOS	33,01
7	GUATEMALA	GARIFUN...	33,54
8	GUATEMALA	LADINOS	33,41
9	GUATEMALA	INDIGENAS	33,05

3.4.2.14. Consulta No. 14

Desplegar el nombre del pais en el cual las elecciones han sido más peleadas. Para determinar esto se debe calcular la diferencia de porcentajes de votos entre el partido que obtuvo más votos y el partido que obtuvo menos votos.

La consulta principal posee una subconsulta. Esta consulta muestra las diferencias de las votaciones para cada pais. Esta consulta utiliza la función max y min. Estas permitirán la resta de la consulta y devolverán un valor donde el más pequeño será la elección más peleada. Esto es devuelto por la consulta principal y lo desplegará. La figura 3.56. muestra esta consulta.

Figura 3.56. Script

```
SELECT lu1.nomblugar as "NOMBRE PAIS ELE + PELEADAS"
FROM
(
        SELECT TABTO paidlugar22 as paidlugar33, MAX(TABTO.totalprimp1)-MIN(TABTO.totalprimp1)
                        as difer
        FROM
        (
                SELECT LUGARES1.paidlugar11 as paidlugar22, vot1.idpolitico as paidpolitico22,
                                SUM(vot1.cantvoto) as totalprimp1
                FROM
                POBLACION pob1,
                (
                        SELECT DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
```

Figura 112. Páginas 137 – 140 Libro “Bases de datos principiantes”


```
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )       LUGARES1, ELECCION ele1, POLITICO pol1,
                VOTACION vot1
                WHERE   vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = pob1.idpoblacion AND
                                vot1.idpolitico = pol1.idpolitico
                GROUP BY LUGARES1.padlugar11, vot1.idpolitico
                ORDER BY LUGARES1.padlugar11, totalprimp1 DESC, vot1.idpolitico
        )       TABTO
        GROUP BY TABTO.padlugar22
        ORDER BY drler ASC
)       TABLON, LUGAR lu1
WHERE   TABLON.padlugar33 = lu1.idlugar1 AND
        rownum < 2
.
```

NOMBRE PAIS ELE + PELEADAS
1 EL SALVADOR

3.4.2.15. Consulta No. 15

Desplegar el número de departamentos, el número de municipios y el total de votos de todos los departamentos, agrupados por todos los departamentos que tienen el mismo número de municipios. Ordenados por número de municipios. (Los datos son solo un ejemplo y no son datos que fueron cargados). Por ejemplo:

departamentos	municipios	votos
5	20	1000
8	12	1500
....	....	...

La consulta principal posee una subconsulta. Esta subconsulta suma la cantidad de votos de los departamentos y además suma la cantidad de municipios. La consulta principal se encarga de sumar la cantidad de

departamentos devuelto por la subconsulta. Esto permite desplegar el resultado. La figura 3.57. muestra esta consulta.

Figura 3.57. Script consulta 15

```
SELECT  count(SUMDEP.iddept) as "TOTAL DEPARTAMENTOS", SUMDEP.totalmunic as
                "TOTAL MUNICIPIOS", SUM(SUMDEP.sumatotalvotos) as "# VOTOS"
FROM
(
        SELECT  SUMMUN.idpais as idpais, SUMMUN.iddept as iddept, COUNT(SUMMUN.idmuni)
                        as totalmunic, SUM(SUMMUN.sumatotalvotos) as sumatotalvotos
        FROM
        (
                SELECT  lu1.idlugar1 as idpais, lu3.idlugar1 as iddept, lu4.idlugar1 as
                                idmuni, SUM(vot1.cantvoto) as sumatotalvotos
                FROM    VOTACION vot1, ELECCION ele1, LUGAR lu1, LUGAR lu2, LUGAR lu3,
                                LUGAR lu4, TIPOLUGAR tp1, TIPOLUGAR tp2, TIPOLUGAR tp3, TIPOLUGAR
tp4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2 AND
                                vot1.ideleccion = ele1.ideleccion AND
                                ele1.idlugar1 = lu4.idlugar1
                GROUP BY lu1.idlugar1, lu3.idlugar1, lu4.idlugar1
                ORDER BY sumatotalvotos DESC
        )       SUMMUN
        GROUP BY SUMMUN.idpais, SUMMUN.iddept
)       SUMDEP
GROUP BY SUMDEP.totalmunic
ORDER BY SUMDEP.totalmunic DESC, SUM(SUMDEP.sumatotalvotos) DESC
.
```

	TOTAL DEPARTAMENTOS	TOTAL MUNICIPIOS	# VOTOS
1	1	33	3573044
2	2	31	6470233
3	3	28	8984755
4	1	26	2848742
5	1	25	2593013
6	1	24	2610170
7	3	23	7463991
8	2	22	4723213
9	2	21	4567180
10	2	20	4396233

3.4.2.16. Consulta No. 16

Desplegar el nombre del pais, el nombre del municipio de todos los municipios cuyo porcentaje de votos de escolaridad primaria sea mayor a un porcentaje dado. Donde el ciento por ciento es el total de votos de escolaridad (primaria, medio y universitario).

La consulta principal posee una subconsulta. Esta subconsulta se divide en dos subconsultas. La primera devuelve el total de votos de alfabetos por pais. La segunda subconsulta permite devolver el total de alfabetos por municipio. Esta consulta resulta de la unión de tres subconsultas que permiten encontrar la escolaridad de los votantes. La consulta principal se encarga de elegir el rango que determina que tipo de porcentaje devolverá la consulta. La figura 3.58. muestra esta consulta.

Figura 3.58. Script consulta 16

```
SELECT  TABLOU.nomblugarp as "NOMBRE PAIS", TABLOU.nomblugars as "NOMBRE MUNI",
                TABLOU.porprimus as "%DE PRIMARIA"
FROM
(
        SELECT  lug1.nomblugar as nomblugarp, lug2.nomblugar as nomblugars,
                        round((VOTEDUM.totalprimp1*100)/VOTEMU.totalvotmu,2) as porprimus
        FROM
        (
                SELECT  LUGARES1.padlugar11 as padlugar22, LUGARES1.muidlugar11 as
                                muidlugar22, SUM(vot1.cantvoto) as totalprimp1
                FROM
                (
                        SELECT  pob1.idpoblacion as idpoblacion1
                        FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                        WHERE   atr1.nombatributo like '%PRIMARIA%' AND
                                        car1.idatributo = atr1.idatributo AND
                                        pob1.idpoblacion = car1.idpoblacion
                )       POBPRIM1,
                (
                        SELECT  DISTINCT lu1.idlugar1 as padlugar11, lu2.idlugar1 as
                                        reidlugar11, lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE   tp1.nombtipolugar like '%Pais%' AND
                                        lu1.idtipolugar = tp1.idtipolugar AND
                                        tp2.nombtipolugar like '%Region%' AND
                                        lu2.idtipolugar = tp2.idtipolugar AND
                                        tp3.nombtipolugar like '%Departamento%' AND
                                        lu3.idtipolugar = tp3.idtipolugar AND
                                        tp4.nombtipolugar like '%Municipio%' AND
                                        lu4.idtipolugar = tp4.idtipolugar AND
                                        lu1.idlugar1 = lu2.idlugar2 AND
                                        lu2.idlugar1 = lu3.idlugar2 AND
                                        lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )       LUGARES1, ELECCION ele1,
                VOTACION vot1
                WHERE   vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = POBPRIM1.idpoblacion1
                GROUP BY LUGARES1.padlugar11,  LUGARES1.muidlugar11
        )       VOTEDUM,
```



```
(
SELECT  VOTEDU.padlugar22 as padlugar33,  VOTEDU.muidlugar22 as muidlugar33,
                SUM(VOTEDU.totalprimp1) as totalvotmu
FROM
(
        SELECT  LUGARES1.padlugar11 as padlugar22, LUGARES1.muidlugar11 as
                        muidlugar22, SUM(vot1.cantvoto) as totalprimp1, 'PRIMARIA' as nivel
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE   atr1.nombatributo like '%PRIMARIA%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )       POBPRIM1,
        (
                SELECT  DISTINCT lu1.idlugar1 as padlugar11, lu2.idlugar1 as
                                reidlugar11, lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE   vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBPRIM1.idpoblacion1
        GROUP BY LUGARES1.padlugar11,  LUGARES1.muidlugar11
UNION
SELECT  LUGARES1.padlugar11 as padlugar22, LUGARES1.muidlugar11 as muidlugar22,
                SUM(vot1.cantvoto) as totalprimp1, 'NIVEL MEDIO' as nivel
FROM
(
        SELECT  pob1.idpoblacion as idpoblacion1
        FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
        WHERE   atr1.nombatributo like '%NIVEL MEDIO%' AND
                        car1.idatributo = atr1.idatributo AND
                        pob1.idpoblacion = car1.idpoblacion
)       POBPRIM1,
(
        SELECT  DISTINCT lu1.idlugar1 as padlugar11, lu2.idlugar1 as reidlugar11,
                        lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
        WHERE   tp1.nombtipolugar like '%Pais%' AND
                        lu1.idtipolugar = tp1.idtipolugar AND
                        tp2.nombtipolugar like '%Region%' AND
                        lu2.idtipolugar = tp2.idtipolugar AND
                        tp3.nombtipolugar like '%Departamento%' AND
                        lu3.idtipolugar = tp3.idtipolugar AND
                        tp4.nombtipolugar like '%Municipio%' AND
                        lu4.idtipolugar = tp4.idtipolugar AND
                        lu1.idlugar1 = lu2.idlugar2 AND
                        lu2.idlugar1 = lu3.idlugar2 AND
                        lu3.idlugar1 = lu4.idlugar2
```

Figura 113. Páginas 141 – 144 Libro “Bases de datos principiantes”

```
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE   vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBPRIM1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11,  LUGARES1.muidlugar11
        UNION
        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as muidlugar22,
                        SUM(vot1.cantvoto) as totalprimp1, 'UNIVERSITARIOS' as nivel
        FROM
        (
                SELECT  pob1.idpoblacion as idpoblacion1
                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                WHERE   atr1.nombatributo like '%UNIVERSITARIOS%' AND
                                car1.idatributo = atr1.idatributo AND
                                pob1.idpoblacion = car1.idpoblacion
        )       POBPRIM1,
        (
                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
        )       LUGARES1, ELECCION ele1,
        VOTACION vot1
        WHERE   vot1.ideleccion = ele1.ideleccion AND
                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                        vot1.idpoblacion = POBPRIM1.idpoblacion1
        GROUP BY LUGARES1.paidlugar11,  LUGARES1.muidlugar11
)       VOTEDU
GROUP BY VOTEDU.paidlugar22, VOTEDU.muidlugar22
ORDER BY VOTEDU.paidlugar22, VOTEDU.muidlugar22
)       VOTEMU,         LUGAR lug1, LUGAR lug2
WHERE   VOTEDUM.paidlugar22 = VOTEMU.paidlugar33 AND
                VOTEDUM.muidlugar22 = VOTEMU.muidlugar33 AND
                VOTEDUM.paidlugar22 = lug1.idlugar1 AND
                VOTEDUM.muidlugar22 = lug2.idlugar1
        ORDER BY porprimus desc, lug1.nomblugar, lug2.nomblugar
) TABLOU
WHERE   TABLOU.porprimus <= 100 AND/*rango inferior del porcentaje*/
                TABLOU.porprimus > 0/*rango superior del porcentaje*/
.
```


	NOMBRE PAIS	NOMBRE MUNI	%DE PRIMARIA
1	GUATEMALA	Sololá	49,8
2	HONDURAS	San Francisco de Ojuera	44,81
3	Panama	Emberá-Wounaan.	44,64
4	Nicaragua	Belén	44,1
5	HONDURAS	La Jigua	43,33
6	GUATEMALA	San Benito	42,9
7	GUATEMALA	Tacaná	42,86
8	EL SALVAD...	El Paraiso	42,83
9	GUATEMALA	Malacatancito	42,83

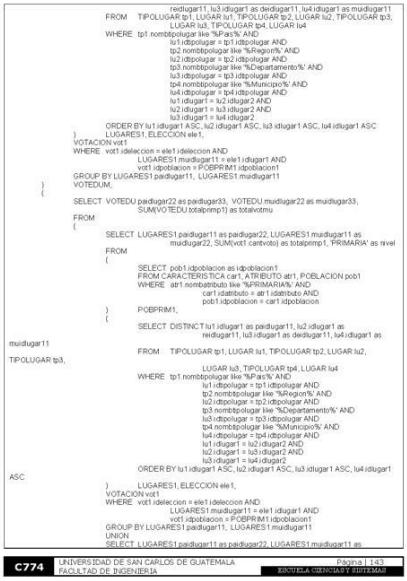
3.4.2.17. Consulta No. 17

Desplegar el nombre del pais, nombre del municipio de todos los municipios cuyo porcentaje de votantes analfabetos sea igual o superior a un porcentaje dado.

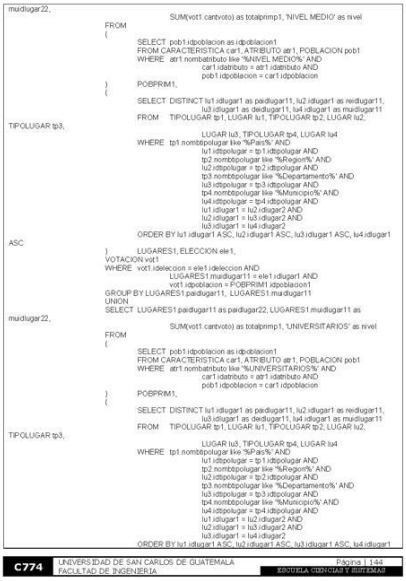
La consulta principal se integra por una subconsulta. Esta subconsulta se divide en dos subconsultas. La primer subconsulta despliega el total de votos de analfabetos. La segunda subconsulta despliega el total de votos tanto de alfabetos como de analfabetos. La consulta principal devuelve el porcentaje de los analfabetos. Esto permite que esta consulta se establezca el porcentaje de la cantidad de analfabetos que demostrará el resultado de la consulta. La figura 3.59. muestra esta consulta.

Figura 3.59. Script consulta 17

```
SELECT  TABLOU.nomblugarp as "NOMBRE PAIS", TABLOU.nomblugars as "NOMBRE MUNI",
                TABLOU.porprimus as "%DE ANALFABETOS"
FROM
(
        SELECT lug1.nomblugar as nomblugarp, lug2.nomblugar as nomblugars,
                        round((VOTEDUM.totalprimp1*100)/VOTEMU.totalvotmu,2) as porprimus
        FROM
        (
                SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
                                muidlugar22, SUM(vot1.cantvoto) as totalprimp1
                FROM
                (
                        SELECT  pob1.idpoblacion as idpoblacion1
                        FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                        WHERE   atr1.nombatributo like '%ANALFABETOS%' AND
                                        car1.idatributo = atr1.idatributo AND
                                        pob1.idpoblacion = car1.idpoblacion
                )       POBPRIM1,
                (
                        SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as
```

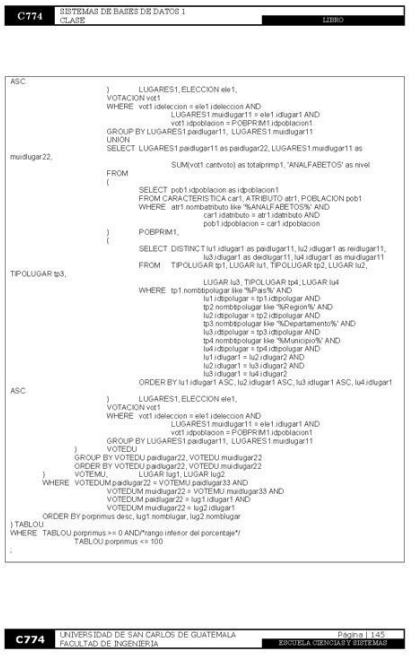



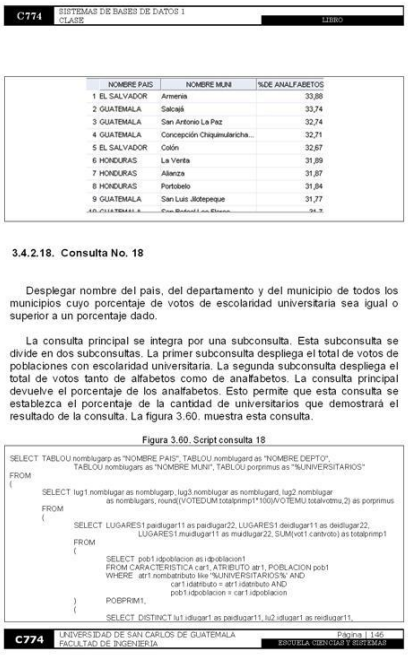
```
                                reidlugar11, lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE   tp1.nombtipolugar like '%Pais%' AND
                                        lu1.idtipolugar = tp1.idtipolugar AND
                                        tp2.nombtipolugar like '%Region%' AND
                                        lu2.idtipolugar = tp2.idtipolugar AND
                                        tp3.nombtipolugar like '%Departamento%' AND
                                        lu3.idtipolugar = tp3.idtipolugar AND
                                        tp4.nombtipolugar like '%Municipio%' AND
                                        lu4.idtipolugar = tp4.idtipolugar AND
                                        lu1.idlugar1 = lu2.idlugar2 AND
                                        lu2.idlugar1 = lu3.idlugar2 AND
                                        lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1 ASC
                )       LUGARES1, ELECCION ele1,
                VOTACION vot1
                WHERE   vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = POBPRIM1.idpoblacion1
        )       VOTEDUM,
        (
                SELECT  VOTEDU.paidlugar22 as paidlugar33,  VOTEDU.muidlugar22 as muidlugar33,
                                SUM(VOTEDU.totalprimp1) as totalvotmu
                FROM
                (
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
                                        muidlugar22, SUM(vot1.cantvoto) as totalprimp1, 'PRIMARIA' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE   atr1.nombatributo like '%PRIMARIA%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as
                                                reidlugar11, lu3.idlugar1 as deidlugar11, lu4.idlugar1 as
muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE   vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11,  LUGARES1.muidlugar11
                        UNION
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
```

```
muidlugar22,
                                SUM(vot1.cantvoto) as totalprimp1, 'NIVEL MEDIO' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE   atr1.nombatributo like '%NIVEL MEDIO%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE   vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11,  LUGARES1.muidlugar11
                        UNION
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
muidlugar22,
                                SUM(vot1.cantvoto) as totalprimp1, 'UNIVERSITARIOS' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE   atr1.nombatributo like '%UNIVERSITARIOS%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE   tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
```

Figura 114. Páginas 145 – 148 Libro “Bases de datos principiantes”





	NOMBRE PAIS	NOMBRE MUNI	%DE ANALFABETOS
1	EL SALVADOR	Armenia	33,88
2	GUATEMALA	Salcajá	33,74
3	GUATEMALA	San Antonio La Paz	32,74
4	GUATEMALA	Concepción Chiquirichapa...	32,71
5	EL SALVADOR	Colón	32,67
6	HONDURAS	La Venta	31,89
7	HONDURAS	Alianza	31,87
8	HONDURAS	Portobelo	31,84
9	GUATEMALA	San Luis Jilotepeque	31,77

3.4.2.18. Consulta No. 18

Desplegar nombre del pais, del departamento y del municipio de todos los municipios cuyo porcentaje de votos de escolaridad universitaria sea igual o superior a un porcentaje dado.

La consulta principal se integra por una subconsulta. Esta subconsulta se divide en dos subconsultas. La primer subconsulta despliega el total de votos de poblaciones con escolaridad universitaria. La segunda subconsulta despliega el total de votos tanto de alfabetos como de analfabetos. La consulta principal devuelve el porcentaje de los analfabetos. Esto permite que esta consulta se establezca el porcentaje de la cantidad de universitarios que demostrará el resultado de la consulta. La figura 3.60. muestra esta consulta.

Figura 3.60. Script consulta 18

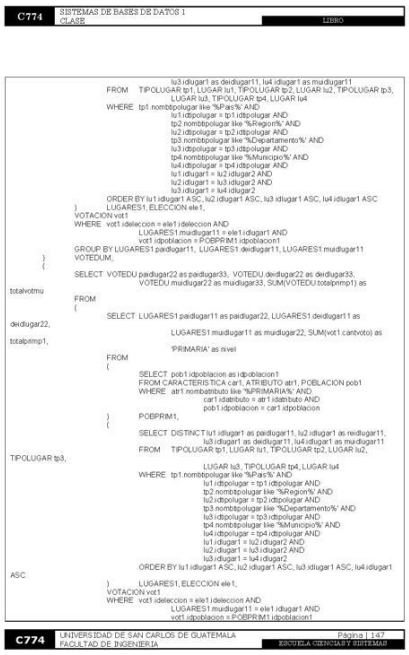


Figura 115. Páginas 149 – 152 Libro “Bases de datos principiantes”

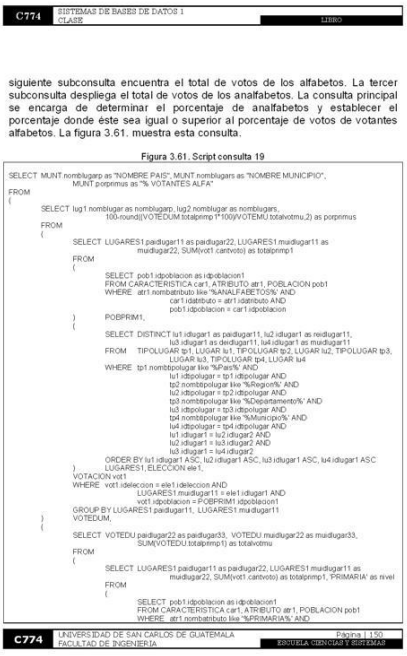

```
                                    lu3.idtipolugar = tp3.idtipolugar AND
                                    tp4.nombtipolugar like '%Municipio%' AND
                                    lu4.idtipolugar = tp4.idtipolugar AND
                                    lu1.idlugar1 = lu2.idlugar2 AND
                                    lu2.idlugar1 = lu3.idlugar2 AND
                                    lu3.idlugar1 = lu4.idlugar2
                            ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                    )       LUGARES1, ELECCION ele1,
                    VOTACION vot1
                    WHERE   vot1.ideleccion = ele1.ideleccion AND
                                    LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                    vot1.idpoblacion = POBPRIM1.idpoblacion1
                    GROUP BY LUGARES1.paidlugar11, LUGARES1.deidlugar11, LUGARES1.muidlugar11
            )       VOTEDU
            GROUP BY VOTEDU.paidlugar22, VOTEDU.deidlugar22, VOTEDU.muidlugar22
            ORDER BY VOTEDU.paidlugar22, VOTEDU.deidlugar22, VOTEDU.muidlugar22
    )       VOTEMU,         LUGAR lug1, LUGAR lug2, LUGAR lug3
    WHERE   VOTEDUM.paidlugar22 = VOTEMU.paidlugar33 AND
                    VOTEDUM.deidlugar22 = VOTEMU.deidlugar33 AND
                    VOTEDUM.muidlugar22 = VOTEMU.muidlugar33 AND
                    VOTEDUM.paidlugar22 = lug1.idlugar1 AND
                    VOTEDUM.muidlugar22 = lug2.idlugar1 AND
                    VOTEDUM.deidlugar22 = lug3.idlugar1
    ORDER BY porprimus desc, lug1.nomblugar, lug2.nomblugar
) TABLOU
WHERE   TABLOU.porprimus >= 0 AND/*rango inferior del porcentaje*/
                TABLOU.porprimus <= 100
;
```

	NOMBRE PAIS	NOMBRE DEPTO	NOMBRE MUNI	%UNIVERSITARIOS
1	GUATEMALA	San Marcos	San Pedro	46,85
2	EL SALVADOR	Chalatenango	Chalatenango	46,42
3	GUATEMALA	Jutiapa	Asunción Mita	46,09
4	GUATEMALA	Guatemala	Villa Canales	45,12
5	HONDURAS	Copán	San José	44,97
6	Costa Rica	Cartago	Oreamuno	44,89
7	GUATEMALA	Suchitepequez	Pueblo Nuevo	44,34
8	HONDURAS	Lempira	Tambla	43,97
9	EL SALVADOR	San Miguel	El Tránsito	43,06

3.4.2.19. Consulta No. 19

Desplegar todos los municipios cuyo porcentaje de votos recibidos de votantes alfabetos sea igual o superior al porcentaje de votos de votantes alfabetos en todos los países, es decir, en todos los votos recibidos en todos los países.

La consulta principal se divide en tres subconsultas. La primer subconsulta devuelve el total de votos de todas las poblaciones (alfabetos y analfabetos). La



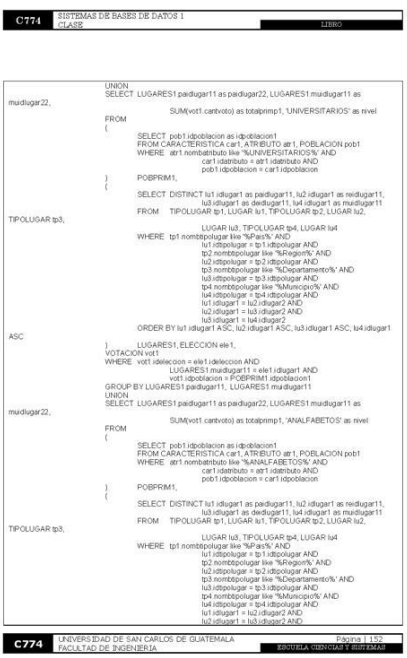
siguiente subconsulta encuentra el total de votos de los alfabetos. La tercer subconsulta despliega el total de votos de los analfabetos. La consulta principal se encarga de determinar el porcentaje de analfabetos y establecer el porcentaje donde éste sea igual o superior al porcentaje de votos de votantes alfabetos. La figura 3.61. muestra esta consulta.

Figura 3.61. Script consulta 19

```
SELECT  MUNT.nomblugarp as "NOMBRE PAIS", MUNT.nomblugars as "NOMBRE MUNICIPIO",
                MUNT.porprimus as "% VOTANTES ALFA"
FROM
(
        SELECT lug1.nomblugar as nomblugarp, lug2.nomblugar as nomblugars,
                        100-round((VOTEDUM.totalprimp1*100)/VOTEMU.totalvotmu,2) as porprimus
        FROM
        (
                SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
                                muidlugar22, SUM(vot1.cantvoto) as totalprimp1
                FROM
                (
                        SELECT  pob1.idpoblacion as idpoblacion1
                        FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                        WHERE  atr1.nombatributo like '%ANALFABETOS%' AND
                                        car1.idatributo = atr1.idatributo AND
                                        pob1.idpoblacion = car1.idpoblacion
                )       POBPRIM1,
                (
                        SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                        lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2, TIPOLUGAR tp3,
                                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                        lu1.idtipolugar = tp1.idtipolugar AND
                                        tp2.nombtipolugar like '%Region%' AND
                                        lu2.idtipolugar = tp2.idtipolugar AND
                                        tp3.nombtipolugar like '%Departamento%' AND
                                        lu3.idtipolugar = tp3.idtipolugar AND
                                        tp4.nombtipolugar like '%Municipio%' AND
                                        lu4.idtipolugar = tp4.idtipolugar AND
                                        lu1.idlugar1 = lu2.idlugar2 AND
                                        lu2.idlugar1 = lu3.idlugar2 AND
                                        lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                )       LUGARES1, ELECCION ele1,
                VOTACION vot1
                WHERE  vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = POBPRIM1.idpoblacion1
                GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11
        )       VOTEDUM,
        (
                SELECT  VOTEDU.paidlugar22 as paidlugar33,  VOTEDU.muidlugar22 as muidlugar33,
                                SUM(VOTEDU.totalprimp1) as totalvotmu
                FROM
                (
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
                                        muidlugar22, SUM(vot1.cantvoto) as totalprimp1, 'PRIMARIA' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%PRIMARIA%' AND
```



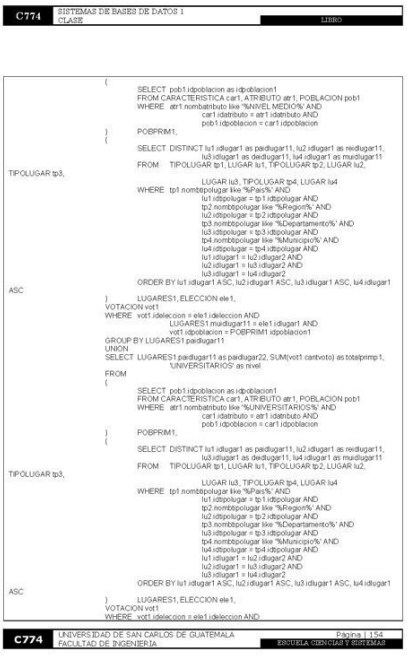
```
                                        car1.idatributo = atr1.idatributo AND
                                        pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11
                        UNION
                        SELECT LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
muidlugar22,
                                        SUM(vot1.cantvoto) as totalprimp1, 'NIVEL MEDIO' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%NIVEL MEDIO%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11
```

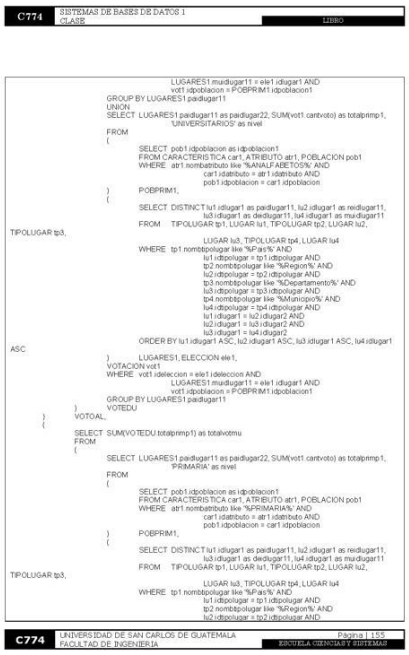

```
                        UNION
                        SELECT LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
muidlugar22,
                                        SUM(vot1.cantvoto) as totalprimp1, 'UNIVERSITARIOS' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%UNIVERSITARIOS%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE  vot1.ideleccon = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11, LUGARES1.muidlugar11
                        UNION
                        SELECT LUGARES1.paidlugar11 as paidlugar22, LUGARES1.muidlugar11 as
muidlugar22,
                                        SUM(vot1.cantvoto) as totalprimp1, 'ANALFABETOS' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%ANALFABETOS%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as deidlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
```

Figura 116. Páginas 153 – 156 Libro “Bases de datos principiantes”

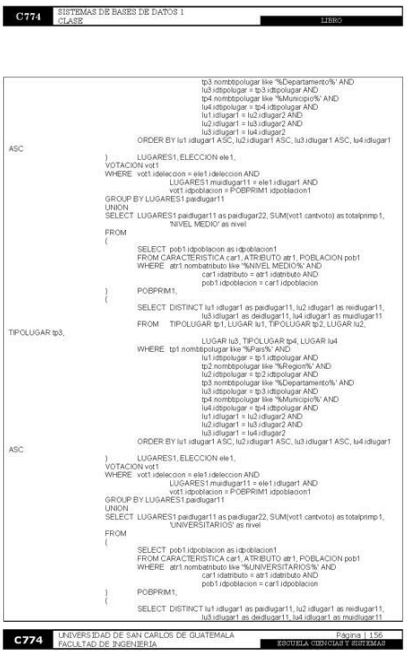
```
                                        lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11,  LUGARES1.muidlugar11
                )       VOTEDU
                GROUP BY VOTEDU.paidlugar22, VOTEDU.muidlugar22
                ORDER BY VOTEDU.paidlugar22, VOTEDU.muidlugar22
        )       VOTEMU,         LUGAR lug1, LUGAR lug2
        WHERE  VOTEDUM.paidlugar22 = VOTEMU.paidlugar33 AND
                VOTEDUM.muidlugar22 = VOTEMU.muidlugar33 AND
                VOTEDUM.paidlugar22 = lug1.idlugar1 AND
                VOTEDUM.muidlugar22 = lug2.idlugar1
        ORDER BY porprimus desc, lug1.nomblugar, lug2.nomblugar
) MUNT,
(
        SELECT round((VOTOTO.totalvotmu*100)/VOTOAL.totalvotmu,2) AS portotal
        FROM
        (
                SELECT  SUM(VOTEDU.totalprimp1) as totalvotmu
                FROM
                (
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                                        'PRIMARIA' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%PRIMARIA%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as dedlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11
                        UNION
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                                        'NIVEL MEDIO' as nivel
                        FROM
```

```
                (
                        SELECT  pob1.idpoblacion as idpoblacion1
                        FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                        WHERE  atr1.nombatributo like '%NIVEL MEDIO%' AND
                                        car1.idatributo = atr1.idatributo AND
                                        pob1.idpoblacion = car1.idpoblacion
                )       POBPRIM1,
                (
                        SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                        lu3.idlugar1 as dedlugar11, lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                        lu1.idtipolugar = tp1.idtipolugar AND
                                        tp2.nombtipolugar like '%Region%' AND
                                        lu2.idtipolugar = tp2.idtipolugar AND
                                        tp3.nombtipolugar like '%Departamento%' AND
                                        lu3.idtipolugar = tp3.idtipolugar AND
                                        tp4.nombtipolugar like '%Municipio%' AND
                                        lu4.idtipolugar = tp4.idtipolugar AND
                                        lu1.idlugar1 = lu2.idlugar2 AND
                                        lu2.idlugar1 = lu3.idlugar2 AND
                                        lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                )       LUGARES1, ELECCION ele1,
                VOTACION vot1
                WHERE  vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = POBPRIM1.idpoblacion1
                GROUP BY LUGARES1.paidlugar11
                UNION
                SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                                'UNIVERSITARIOS' as nivel
                FROM
                (
                        SELECT  pob1.idpoblacion as idpoblacion1
                        FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                        WHERE  atr1.nombatributo like '%UNIVERSITARIOS%' AND
                                        car1.idatributo = atr1.idatributo AND
                                        pob1.idpoblacion = car1.idpoblacion
                )       POBPRIM1,
                (
                        SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                        lu3.idlugar1 as dedlugar11, lu4.idlugar1 as muidlugar11
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                        LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE  tp1.nombtipolugar like '%Pais%' AND
                                        lu1.idtipolugar = tp1.idtipolugar AND
                                        tp2.nombtipolugar like '%Region%' AND
                                        lu2.idtipolugar = tp2.idtipolugar AND
                                        tp3.nombtipolugar like '%Departamento%' AND
                                        lu3.idtipolugar = tp3.idtipolugar AND
                                        tp4.nombtipolugar like '%Municipio%' AND
                                        lu4.idtipolugar = tp4.idtipolugar AND
                                        lu1.idlugar1 = lu2.idlugar2 AND
                                        lu2.idlugar1 = lu3.idlugar2 AND
                                        lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                )       LUGARES1, ELECCION ele1,
                VOTACION vot1
                WHERE  vot1.ideleccion = ele1.ideleccion AND
```

```
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11
                        UNION
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                                        'UNIVERSITARIOS' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%ANALFABETO%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as dedlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11
                )       VOTEDU
        )       VOTOAL,
        (
                SELECT  SUM(VOTEDU.totalprimp1) as totalvotmu
                FROM
                (
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                                        'PRIMARIA' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%PRIMARIA%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as dedlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
```

```
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11
                        UNION
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                                        'NIVEL MEDIO' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%NIVEL MEDIO%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as dedlugar11, lu4.idlugar1 as muidlugar11
                                FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                                WHERE  tp1.nombtipolugar like '%Pais%' AND
                                                lu1.idtipolugar = tp1.idtipolugar AND
                                                tp2.nombtipolugar like '%Region%' AND
                                                lu2.idtipolugar = tp2.idtipolugar AND
                                                tp3.nombtipolugar like '%Departamento%' AND
                                                lu3.idtipolugar = tp3.idtipolugar AND
                                                tp4.nombtipolugar like '%Municipio%' AND
                                                lu4.idtipolugar = tp4.idtipolugar AND
                                                lu1.idlugar1 = lu2.idlugar2 AND
                                                lu2.idlugar1 = lu3.idlugar2 AND
                                                lu3.idlugar1 = lu4.idlugar2
                                ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                        )       LUGARES1, ELECCION ele1,
                        VOTACION vot1
                        WHERE  vot1.ideleccion = ele1.ideleccion AND
                                        LUGARES1.muidlugar11 = ele1.idlugar1 AND
                                        vot1.idpoblacion = POBPRIM1.idpoblacion1
                        GROUP BY LUGARES1.paidlugar11
                        UNION
                        SELECT  LUGARES1.paidlugar11 as paidlugar22, SUM(vot1.cantvoto) as totalprimp1,
                                        'UNIVERSITARIOS' as nivel
                        FROM
                        (
                                SELECT  pob1.idpoblacion as idpoblacion1
                                FROM CARACTERISTICA car1, ATRIBUTO atr1, POBLACION pob1
                                WHERE  atr1.nombatributo like '%UNIVERSITARIOS%' AND
                                                car1.idatributo = atr1.idatributo AND
                                                pob1.idpoblacion = car1.idpoblacion
                        )       POBPRIM1,
                        (
                                SELECT  DISTINCT lu1.idlugar1 as paidlugar11, lu2.idlugar1 as reidlugar11,
                                                lu3.idlugar1 as dedlugar11, lu4.idlugar1 as muidlugar11
```

Figura 117. Páginas 157 – 158 Libro “Bases de datos principiantes”


```
                        FROM    TIPOLUGAR tp1, LUGAR lu1, TIPOLUGAR tp2, LUGAR lu2,
TIPOLUGAR tp3,
                                LUGAR lu3, TIPOLUGAR tp4, LUGAR lu4
                        WHERE   tp1.nombtipolugar like '%Pais%' AND
                                lu1.idtipolugar = tp1.idtipolugar AND
                                tp2.nombtipolugar like '%Region%' AND
                                lu2.idtipolugar = tp2.idtipolugar AND
                                tp3.nombtipolugar like '%Departamento%' AND
                                lu3.idtipolugar = tp3.idtipolugar AND
                                tp4.nombtipolugar like '%Municipio%' AND
                                lu4.idtipolugar = tp4.idtipolugar AND
                                lu1.idlugar1 = lu2.idlugar2 AND
                                lu2.idlugar1 = lu3.idlugar2 AND
                                lu3.idlugar1 = lu4.idlugar2
                        ORDER BY lu1.idlugar1 ASC, lu2.idlugar1 ASC, lu3.idlugar1 ASC, lu4.idlugar1
ASC
                )       LUGARES1, ELECCION ele1;
                VOTACION vot1
                WHERE   vot1.ideleccion = ele1.ideleccion AND
                                LUGARES1.mxidlugar11 = ele1.idlugar1 AND
                                vot1.idpoblacion = POBPRIM1.idpoblacion1
                GROUP BY LUGARES1.paridlugar11
                )       VOTEDU
        )       VOTOTO
)       TOMUN
WHERE   MUNT.porprimus >= TOMUN.portotal
ORDER BY MUNT.porprimus DESC, MUNT.nomblugarp, MUNT.nomblugar1
;
```

	NOMBRE PAIS	NOMBRE MUNICIPIO	% VOTANTES ALFA
1	Costa Rica	Liberia	84,8
2	HONDURAS	El Paraiso	83,57
3	EL SALVADOR	Nueva San Salvador	83,17
4	HONDURAS	San Juan de Flores	82,76
5	HONDURAS	Dulce Nombre de Culmi	82,61
6	HONDURAS	Comayagua	82,49
7	Panama	Boquerón	82,41
8	HONDURAS	Roatan	82,2
9	Nicaragua	Jalapa	82,17

3.4.2.20. Consulta No. 20

Desplegar el nombre del pais, el número total de votos y el porcentaje de votos recibidos por pais para todas las elecciones en todos los paises.

La consulta principal se divide en dos subconsultas. La primera subconsulta permite calcular el total de votos de todas las elecciones por pais. La segunda subconsulta permite calcular el total de votos de todas las elecciones. La

consulta principal entonces permite calcular el porcentaje de votos por pais para todas las elecciones. La figura 3.62. muestra esta consulta.

Figura 3.62. Script consulta 20

```
SELECT TABTO.nombpais as "NOMBRE PAIS",
                round((TABTO.sumatotalvotos*100)/TABTA.sumatotalvotos,2) as "% VOTACION",
                TABTO.sumatotalvotos as "# VOTOS C/PAIS", TABTA.sumatotalvotos as "# VOTOS TOTALES"
FROM
(
        SELECT  lu1.nomblugar as nombpais, SUM(vot1.cantvoto) as sumatotalvotos
        FROM    VOTACION vot1,ELECCION ele1, LUGAR lu1, LUGAR lu2, LUGAR lu3, LUGAR lu4,
                        TIPOLUGAR tp1, TIPOLUGAR tp2, TIPOLUGAR tp3, TIPOLUGAR tp4
        WHERE  tp1.nombtipolugar like '%Pais%' AND
                        lu1.idtipolugar = tp1.idtipolugar AND
                        tp2.nombtipolugar like '%Region%' AND
                        lu2.idtipolugar = tp2.idtipolugar AND
                        tp3.nombtipolugar like '%Departamento%' AND
                        lu3.idtipolugar = tp3.idtipolugar AND
                        tp4.nombtipolugar like '%Municipio%' AND
                        lu4.idtipolugar = tp4.idtipolugar AND
                        lu1.idlugar1 = lu2.idlugar2 AND
                        lu2.idlugar1 = lu3.idlugar2 AND
                        lu3.idlugar1 = lu4.idlugar2 AND
                        vot1.ideleccion = ele1.ideleccion AND
                        ele1.idlugar1 = lu4.idlugar1
        GROUP BY lu1.nomblugar
        ORDER BY sumatotalvotos DESC
)       TABTO,
(
        SELECT  SUM(vot1.cantvoto) as sumatotalvotos
        FROM    VOTACION vot1
)       TABTA
;
```

	NOMBRE PAIS	% VOTACION	# VOTOS C/PAIS	# VOTOS TOTALES
1	GUATEMALA	28,49	35865357	125898088
2	HONDURAS	24,54	30900834	125898088
3	EL SALVADOR	21,53	27102545	125898088
4	Nicaragua	12,97	16334618	125898088
5	Panama	7,2	9064385	125898088
6	Costa Rica	5,27	6630349	125898088

4. APENDICE

CONCLUSIONES

1. Los *Instructor Guideline* de los cursos de Introducción a la Programación y Computación 1, Introducción a la programación 2 y Redes de Nueva Generación, fueron realizados conforme a los contenidos actuales propuestos y aprobados por cada uno de los catedráticos titulares de los cursos.

2. El material visual fue realizado conforme a diapositivas. El contenido de las mismas equivale a los puntos importantes encontrados dentro de los *Instructor Guideline* de cada uno de los cursos. Estos servirán de guía para poder impartir el curso.

3. El *Hand Book* fue realizado de una manera estructurada en el cual se definieron los ejercicios, prácticas preparatorias para los cursos de IPC1, IPC2 y NGN. El *Hand Book* de NGN, además, incluye una sección correspondiente a los proyectos del curso.

4. La documentación de apoyo para los cursos de IPC1 e IPC2 fue realizada conforme a todos los temas contenidos en el IG de ambos cursos. La documentación de apoyo de Sistemas de Bases de Datos se ha orientado de tal forma que sea un libro práctico. Esto quiere decir que se orienta a brindar a los estudiantes la capacidad para la creación de modelos entidad relación y poder realizar consultas sobre esos modelos.

5. La estructuración de los exámenes de los cursos permite que sean tomados como referencia para futuros exámenes. Esto significa que los catedráticos nuevos podrán realizar exámenes acordes al nivel exigido por el curso.

RECOMENDACIONES

1. El Director de la Escuela de Ingeniería en Ciencias y Sistemas debe nombrar encargados de velar para que los contenidos de los IG, *Hand Book*, material visual y documentación de apoyo se mantengan constantemente actualizados. Estas personas deben colaborar con un catedrático titular para poder discutir los cambios necesarios. Esta actualización se puede realizar preferiblemente en un período de seis meses o de un año.

2. Las actualizaciones realizadas a los IG, *Hand Book*, material visual y documentación de apoyo deben poder ser administradas. Esto quiere decir que debe llevarse un control de versiones de los nuevos documentos, generado a partir del primer cambio que sufran luego de su publicación.

BIBLIOGRAFÍA

1. Craig, Larman. UML Y Patrones, Introducción al Análisis y Diseño Orientado a Objetos. Editorial Prentice Hall. 2ª. Edición. México. Año 2004.

2. Date, C. J. Introducción a los sistemas de bases de datos. Editorial Prentice Hall. 7ª. Edición. México. Año 2001.

3. Joyanes, L. y Zahonero, I. Programación en Java 2 (algoritmos, estructura de datos y programación orientada a objetos). Editorial McGraw-Hill / Interamericana de España. 2ª. Edición. España. Año 2002.

4. Mrutunjaya, Panda. Curso de Planificación Educativa a través de *Instructor Guidelines*. India-Guatemala *IT Education Center of Excellence*. Guatemala. Año 2008.

5. Zuidweg, Johan. *Next Generation Intelligent Networks* (*Artech House Telecommunications Library*). Editorial Artech House. 1ª Edición. Reino Unido. Año 2002.

ANEXOS

Los *Instructor Guideline*, los *Hand Book*, el material visual y la documentación de apoyo ha sido elaborada bajo supervisión y soporte de los ingenieros responsables de cada curso pertenecientes a la Escuela de Ciencias y Sistemas, quienes fueron asignados por el Ingeniero Armín Mazariegos, asesor del proyecto.

La sección de anexos presenta seis cartas. Estas cartas pertenecen a los ingenieros que supervisaron el proyecto:

- Ing. Marlon Pérez Turk – Introducción a la Programación y Computación 1.
- Ing. Marlon Pérez Turk – Introducción a la Programación y Computación 2.
- Ing. Luis Espino Barrios – Sistemas de Bases de Datos 1.
- Ing. Pedro Pablo Hernández – Redes de Nueva Generación.

Las cartas citadas corresponden a la aceptación de cada uno de los *Instructor Guideline* y a la documentación de apoyo, donde a criterio de cada ingeniero queda indicado que se ha cumplido con el objetivo y el contenido planteado para cada documento.

Guatemala 01 de julio de 2009

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado el **Instructor Guideline (IG) de la "*Clase de Introducción a la Programación y Computación 1*"** del alumno Mario José Bautista Fuentes con carné 200312942, del Proyecto "*Aplicación de la metodología de la empresa TATA CONSULTANCY SERVICES a las áreas de Programación, Redes y Bases de Datos de la carrera de Ciencias y Sistemas*", y a mi parecer cumple con los objetivos y requisitos necesarios de guía de la clase.

Agradeciendo su atención a la presente,

Atentamente,

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Pérez Turk.
Director de la Escuela de Ciencias y Sistemas
Facultad de Ingeniería, USAC

Guatemala 01 de julio de 2009

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado el **libro de IPC1** ***"Clase de Introducción a la Programación y Computación 1"*** del alumno Mario José Bautista Fuentes con carné 200312942, del Proyecto "*Aplicación de la metodología de la empresa TATA CONSULTANCY SERVICES a las áreas de Programación, Redes y Bases de Datos de la carrera de Ciencias y Sistemas*", y a mi parecer cumple con los objetivos y requisitos necesarios de un libro práctico que puede ser utilizado como apoyo en el curso de Introducción a la Programación y Computación 1.

Agradeciendo su atención a la presente,

Atentamente,

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Pérez Turk.
Director de la Escuela de Ciencias y Sistemas
Facultad de Ingeniería, USAC

Guatemala 01 de julio de 2009

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado el **Instructor Guideline (IG) de la "*Clase de Introducción a la Programación y Computación 2*"** del alumno Mario José Bautista Fuentes con carné 200312942, del Proyecto "*Aplicación de la metodología de la empresa TATA CONSULTANCY SERVICES a las áreas de Programación, Redes y Bases de Datos de la carrera de Ciencias y Sistemas*", y a mi parecer cumple con los objetivos y requisitos necesarios de guía de la clase.

Agradeciendo su atención a la presente,

Atentamente,

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Pérez Turk.
Director de la Escuela de Ciencias y Sistemas
Facultad de Ingeniería, USAC

Guatemala 01 de julio de 2009

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado el **libro de IPC2** ***"Clase de Introducción a la Programación y Computación 2"*** del alumno Mario José Bautista Fuentes con carné 200312942, del Proyecto "*Aplicación de la metodología de la empresa TATA CONSULTANCY SERVICES a las áreas de Programación, Redes y Bases de Datos de la carrera de Ciencias y Sistemas*", y a mi parecer cumple con los objetivos y requisitos necesarios de un libro práctico que puede ser utilizado como apoyo en el curso de Introducción a la Programación y Computación 2.

Agradeciendo su atención a la presente,

Atentamente,

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Pérez Turk.
Director de la Escuela de Ciencias y Sistemas
Facultad de Ingeniería, USAC

Guatemala 29 de mayo de 2009

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado el **libro de Sistemas de Bases de Datos 1 "*Clase de Sistemas de Bases de Datos 1*"** del alumno Mario José Bautista Fuentes con carné 200312942, del Proyecto "*Aplicación de la metodología de la empresa TATA CONSULTANCY SERVICES a las áreas de Programación, Redes y Bases de Datos de la carrera de Ciencias y Sistemas*", y a mi parecer cumple con los objetivos y requisitos necesarios de un libro práctico que puede ser utilizado como apoyo en el curso de Sistemas de Bases de Datos 1.

Agradeciendo su atención a la presente,

Atentamente,

Ing. Luis Espino Barrios.
Catedrático de la Escuela de Ciencias y Sistemas
Facultad de Ingeniería, USAC

Guatemala 25 de abril de 2009

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado la **guía de clase (IG) de "*Clase de Redes de Nueva Generación*"** del alumno Mario José Bautista Fuentes con carné 200312942, del Proyecto "*Aplicación de la metodología de la empresa TATA CONSULTANCY SERVICES a las áreas de Programación, Redes y Bases de Datos de la carrera de Ciencias y Sistemas*", y a mi parecer cumple con los objetivos y requisitos necesarios de guía del curso de Redes de Nueva Generación.

Agradeciendo su atención a la presente,

Atentamente,

Ing. Pedro Pablo Hernández R.
Catedrático de la Escuela de Ciencias y Sistemas
Facultad de Ingeniería, USAC