



UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

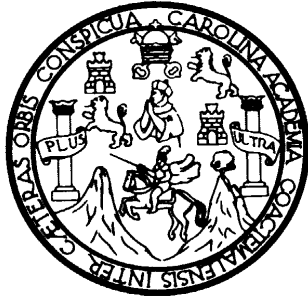
INDEXACIÓN EN BASES DE DATOS
ORIENTADAS A OBJETOS

EDWARD MAURICIO AYAU GODINEZ

ASESORADO POR ING. RENÉ FRANCISCO CONTRERAS QUEMÉ

GUATEMALA, MARZO DE 2004

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**INDEXACIÓN EN BASES DE DATOS
ORIENTADAS A OBJETOS**

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

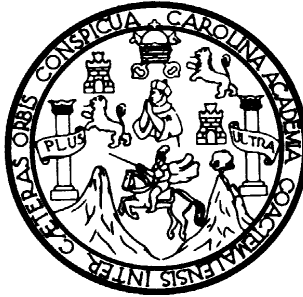
EDWARD MAURICIO AYAU GODINEZ

ASESORADO POR: ING. RENÉ FRANCISCO CONTRERAS QUEMÉ

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, MARZO DE 2004

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Sydney Alexander Samuels Milson
VOCAL I	Ing. Murphy Olympo Paiz Recinos
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Keneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Sydney Alexander Samuels Milson
EXAMINADOR	Ing. Marlon Perez Turk
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
EXAMINADOR	Inga. Virginia Victoria Tala Ayerdi
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

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 graduación titulado:

INDEXACIÓN EN BASES DE DATOS ORIENTADAS A OBJETOS

tema que me fuera asignado por la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería, con fecha 7 de Julio de 2002.

Edward Mauricio Ayau Godinez

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES	VII
GLOSARIO	IX
RESUMEN	XI
OBJETIVOS	XIII
INTRODUCCIÓN	XV
1 LA NECESIDAD EN EL USO DE ÍNDICES	
1.1 La necesidad en el uso de índices	1
1.2 Evolución de los sistemas de indexación	2
1.3 Características del lenguaje de consulta orientado a objetos.....	3
1.3.1 Jerarquías de herencia.....	3
1.3.2 Jerarquías de agregación.....	4
1.3.3 Métodos	4
1.4 Clasificación de las técnicas de indexación.....	4
1.4.1 Estructurales	5
1.4.1.1 Técnicas basadas en jerarquía de herencia	5
1.4.1.2 Técnicas basadas en jerarquía de agregación.....	6
1.4.1.3 Técnicas basadas en jerarquía de herencia y de agregación.....	6
1.4.2 De Comportamiento	7
1.4.2.1 Materialización de métodos.....	8
2 TÉCNICAS DE INDEXACIÓN	
2.1 Técnicas estructurales.....	9
2.1.1 Técnicas de jerarquía de herencia	9
2.1.1.1 <i>Single Class</i> (SC)	9

2.1.1.1.1 Estructura	10
2.1.1.1.2 Operaciones	11
2.1.1.2 CH-Tree	11
2.1.1.2.1 Estructura	11
2.1.1.2.2 Operaciones	13
2.1.1.3 H-Tree	13
2.1.1.3.1 Estructura	14
2.1.1.3.2 Operaciones	15
2.1.1.4 hcC-Tree	15
2.1.1.4.1 Estructura	16
2.1.1.4.2 Operaciones	19
2.1.2 Técnicas de jerarquía de agregación	19
2.1.2.1 <i>Path Index</i> (PX)	21
2.1.2.1.1 Estructura	22
2.1.2.1.2 Operaciones	23
2.1.2.2 <i>Nested</i> (NX)	24
2.1.2.2.1 Estructura	24
2.1.2.2.2 Operaciones	25
2.1.2.3 <i>Multiindex</i> (MX)	26
2.1.2.3.1 Estructura	26
2.1.2.3.2 Operaciones	26
2.1.2.4 <i>Path Dictionary Index</i> (PDI)	27
2.1.2.4.1 Estructura	29
2.1.2.4.2 Operaciones	32
2.1.2.5 <i>Join Index Hierarchy</i> (JIH)	33
2.1.2.5.1 Estructura	35
2.1.2.5.2 Operaciones	38
2.1.3 Técnicas de jerarquía de herencia y agregación	39
2.1.3.1 <i>Nested Inherited</i>	40

2.1.3.1.1	Estructura	40
2.1.3.1.2	Operaciones	41
2.1.3.2	Inherited MultiIndex	42
2.1.3.2.1	Estructura	42
2.1.3.2.2	Operaciones	42
2.2	Técnicas de comportamiento	43
2.2.1	Técnicas basadas en la invocación de métodos	43
2.2.1.1	Materialización de métodos	43
2.2.1.1.1	Estructura	44
2.2.1.1.2	Operaciones	45
3	COMPARACIONES DE EFICIENCIA ENTRE LAS TÉCNICAS DE INDEXACIÓN	
3.1	Técnicas estructurales.....	47
3.1.1	Técnicas de jerarquía de herencia	47
3.1.1.1	<i>Single Class</i> (SC)	47
3.1.1.1.1	Adecuado para consultas en una única clase de la jerarquía	47
3.1.1.2	<i>CH-Tree</i>	48
3.1.1.2.1	Adecuado para consultas que implican varias clases de la jerarquía	48
3.1.1.3	<i>H-Tree</i>	48
3.1.1.3.1	Adecuado para consultas de recuperación en un número pequeño de clases de la jerarquía.....	48
3.1.1.4	<i>hcC-Tree</i>	49
3.1.1.4.1	Es menos dependiente de la distribución de los datos que <i>CH-Tree</i> y <i>H-Tree</i>	49
3.1.1.4.2	Se comporta mejor que el <i>H-Tree</i> cuando el número de clases en la jerarquía aumenta.....	49
3.1.2	Técnicas de jerarquía de agregación	50

3.1.2.1 <i>Path</i> (PX).....	50
3.1.2.1.1 Permite resolver predicados anidados sobre todas las clases del camino.....	50
3.1.2.2 <i>Nested</i> (NX)	50
3.1.2.2.1 Bajo costo en operaciones de recuperación	50
3.1.2.3 <i>Multiindex</i> (MX)	51
3.1.2.3.1 Bajo costo en operaciones de actualización	51
3.1.2.3.2 Alto costo en operaciones de recuperación	51
3.1.2.3.3 PX es el mejor teniendo en cuenta todas las operaciones...51	
3.1.2.3.4 MX requerimientos de espacio intermedios entre NX y PX..52	
3.1.2.4 <i>Path Dictionary Index</i> (PDI).....	52
3.1.2.4.1 Posibilita la existencia de muchos índices sobre atributos...52	
3.1.2.4.2 El PDI tiene mejor rendimiento en la recuperación que el PX	53
3.1.2.4.3 El PX se comporta mejor en consultas de rango	53
3.1.2.4.4 El PD se comporta mejor en actualización que el PX	54
3.1.2.5 <i>Join Index Hierarchy</i> (JIH).....	54
3.1.2.5.1 JIH-C requiere más espacio y más costo de actualización ..54	
3.1.2.5.2 JIH-B se comporta peor que JIH-P y que JIH-C.....	55
3.1.2.5.3 División del esquema de camino para esquemas de caminos largos	55
3.1.3 Técnicas de jerarquía de herencia y agregación	56
3.1.3.1 <i>Nested Inherited</i>	55
3.1.3.1.1 Apropiado cuando todos los atributos del camino son de valor-simple.....	56
3.1.3.2 <i>Inherited MultiIndex</i>	56
3.1.3.2.1 Apropiado cuando las consultas son principalmente sobre la última clase del camino indexado.....	57
3.2 Técnicas de comportamiento	57

3.2.1	Técnicas basadas en la invocación de métodos	57
3.2.1.1	Materialización de métodos	57
3.2.1.1.1	Apropiado para operaciones de búsqueda	58
4	PRUEBAS DE RENDIMIENTO	
4.1	Escenario de pruebas.....	59
4.1.1	Servidor de base de datos relacional	59
4.1.1.1	<i>Hardware</i>	59
4.1.1.2	<i>Software</i>	60
4.1.2	Servidor de base de datos orientada a objetos	60
4.1.2.1	<i>Hardware</i>	60
4.1.2.2	<i>Software</i>	61
4.2	Componentes para la realización de pruebas	61
4.2.1	Base de datos relacional	61
4.2.2	Base de datos orientada a objetos	64
4.2.3	Aplicación de pruebas	67
4.2.3.1	Ingreso y eliminación de información	68
4.2.3.2	Consulta de información	70
4.2.3.2.1	Consulta a una base de datos relacional.....	71
4.2.3.2.2	Consulta a una base de datos orientada a objetos.....	72
4.3	Resultados de las pruebas	73
4.3.1	Pruebas de ingreso	74
4.3.2	Pruebas de consultas	76
4.3.3	Pruebas de eliminación	79
4.4	Análisis de los resultados	81
	CONCLUSIONES	83
	RECOMENDACIONES	85
	BIBLIOGRAFÍA	87

ÍNDICE DE ILUSTRACIONES

FIGURAS

1	Nodo interno y nodo hoja de un árbol B^+	10
2	Nodo hoja de un <i>CH-Tree</i>	12
3	Nodo interno y nodo hoja de un <i>H-Tree</i>	14
4	Nodo interno del <i>hcC-Tree</i>	17
5	Nodo hoja de un <i>hcC-Tree</i>	18
6	Entrada de nodos-IDO del <i>hcC-Tree</i>	18
7	Nodo hoja para un índice PX	22
8	Nodo hoja de un índice <i>Nested</i>	25
9	Índice diccionario de caminos	27
10	Estructura de datos de una s-expresión	30
11	Nodo hoja e interno de un índice <i>identity</i>	31
12	Nodos hoja e interno de un índice <i>attribute</i>	32
13	Camino de esquema de longitud 5	34
14	JIH completa para el camino de esquema de la Figura 13	36
15	JIH base para el camino de esquema de la Figura 13	37
16	JIH parcial para el camino de esquema de la Figura 13	38
17	Formato de la tabla que almacena los métodos materializados	44
18	Modelo entidad relación	62
19	Diagrama de clases	65
20	Forma de ingreso de información a SQL	69
21	Forma de ingreso de información a Oracle	70

22	Forma para consultas a SQL	72
23	Forma para consultas a Oracle.....	73
24	Pruebas de ingreso con índice (Formato HH:MM:SS).....	75
25	Pruebas de ingreso sin índice (Formato HH:MM:SS).....	76
26	Pruebas de consulta con índice (Formato HH:MM:SS)	77
27	Pruebas de consulta sin índice (Formato HH:MM:SS)	78
28	Pruebas de eliminación con índice (Formato HH:MM:SS).....	79
29	Pruebas de eliminación sin índice (Formato HH:MM:SS).....	80

TABLAS

I.	Detalle de muestras	74
II.	Resultado de pruebas de ingreso con índice (Formato HH:MM:SS)	74
III.	Resultado de pruebas de ingreso sin índice (Formato HH:MM:SS).....	75
IV.	Resultado de pruebas de consulta con índice (Formato HH:MM:SS)	77
V.	Resultado de pruebas de consulta sin índice (Formato HH:MM:SS).....	78
VI.	Resultado de pruebas de eliminación con índice (Formato HH:MM:SS).....	79
VII.	Resultado de pruebas de eliminación sin índice (Formato HH:MM:SS).....	80

GLOSARIO

Algoritmo	Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.
Atributo	Es cada una de las cualidades o propiedades de una clase.
Clase	Es la abstracción de un conjunto de objetos de la realidad.
Instancia	Se le llama así a la acción de crear un objeto en base a una clase.
Objeto	Es la forma concreta de una clase.
Método	Son las funciones y/o procedimientos que pueden tener las clases.
Nodo	Es parte más pequeña de un árbol de búsqueda.

RESUMEN

Las bases de datos hoy en día son las más utilizadas para el almacenamiento de la información dentro de cualquier tipo de negocio que sea, ya que proporcionan una gran confiabilidad e integridad de la información y con las bases de datos orientadas a objetos esto se hace más fácil; porque se acoplan al lenguaje de programación.

Ya que la cantidad de información que se maneja es muy grande, se requiere de mecanismos para indexar la información de manera que esta sea fácil y rápida de buscar y obtener. Las técnicas de indexación expuestas muestran la diferencia en la forma en que se tratan los datos dentro de una base de datos relacional y una base de datos orientada a objetos.

Las bases de datos orientadas a objetos almacenan información de una forma muy diferente a las bases de datos relacionales para lo que fue necesario el desarrollo de nuevas técnicas de indexación para los datos. Y estas nuevas técnicas de indexación deben tomar en cuenta tres características que se encuentran impuestas por el modelo de objetos y son Jerarquías de Agregación, Jerarquías de Herencia y Métodos, ya que estas tres características marcan claramente las diferencias con los lenguajes de consulta relacionales.

OBJETIVOS

- **General**

Mostrar las ventajas y desventajas de utilizar la indexación de un sistema de base de datos orientada a objetos con respecto a un sistema relacional.

- **Específicos**

1. Mostrar la clasificación de técnicas de indexación de bases de datos orientadas a objetos
2. Identificación y descripción de los mecanismos de indexación de las bases de datos orientadas a objetos
3. Identificación y descripción de los métodos de recorridos de índices
4. Identificar las diferencias que existen con las técnicas de indexación de las bases de datos relacionales
5. Ventajas ante las técnicas de indexación de las bases de datos relacionales

INTRODUCCIÓN

Los sistemas de bases de datos orientados a objetos fueron necesarios principalmente por el surgimiento de nuevas aplicaciones orientadas al modelo de objetos. Con el surgimiento de estas aplicaciones que representan mejor la semántica de la información fue necesaria una forma de almacenamiento en la que se soportará este tipo de información.

Dado que estas bases de datos orientadas a objetos almacenan información de una forma muy diferente a las bases de datos relacionales, fue necesario el desarrollo de nuevas técnicas de indexación para los datos. Y estas nuevas técnicas de indexación deben de tomar en cuenta tres características que se encuentran impuestas por el modelo de objetos y son:

- Jerarquías de agregación
- Jerarquías de herencia
- Métodos

Estas tres características marcan claramente las diferencias con los lenguajes de consulta relacionales.

Son muchas las técnicas de indexación en el paradigma de orientación a objetos que se han propuesto, por lo tanto lo que se intenta mostrar mediante este tema son las clasificaciones de estas técnicas tomando en cuenta las tres características mencionadas anteriormente.

También se quiere mostrar una comparación del rendimiento en la recuperación y en la actualización de los datos con una base de datos orientada a objetos y una relacional.

1 LA NECESIDAD EN EL USO DE ÍNDICES

1.1 La necesidad en el uso de índices

Un índice, en terminología de base de datos, es una estructura de almacenamiento físico empleada para acelerar el acceso a los datos. Los índices juegan, por tanto, un papel fundamental en las bases de datos orientadas a objetos, de la misma manera que lo hacen en las bases de datos relacionales.

Son un soporte básico e indispensable para acelerar el procesamiento de las consultas, y en definitiva el acceso a los datos, y es precisamente por esto por lo que existe una gran proliferación de dichas técnicas. Dado a que existe gran proliferación de dichas técnicas en el presente trabajo de graduación se describe la forma en la que están clasificadas, por lo que se utilizarán dos capítulos completos para dar una idea más profunda de que tan complejo es minimizar los tiempos de espera en una consulta a la base de datos.

1.2 Evolución de los sistemas de indexación

Dentro de los primeros sistemas de administración de bases de datos se encuentran los sistemas de administración de archivos que realmente eran archivos planos y que cuando se realizaba una actualización o búsqueda, se realizaba la lectura de todo el archivo hasta localizar la información y si el archivo contenía 1000 ó 2000 registros cada vez que se requería información es necesario leer todo el archivo de nuevo y en una aplicación que realice varias búsquedas por segundo esto puede resultar ineficiente.

Luego aparecieron los sistemas de administración de bases de datos relacionales, los cuales solucionaron el problema del acceso a la información. Lo que se realiza con estas bases de datos para agilizar la búsqueda de la información es la creación de otro archivo que contiene información resumida de la información y la referencia para localizar directamente la información, sin necesidad de recorrer todo el archivo plano.

Y por último tenemos los sistemas de administración de bases de datos orientadas a objetos, que combinan las mejores cualidades de los archivos planos, las bases jerárquicas y relacionales. Las bases de datos orientadas a objetos representan el siguiente paso en la evolución de las bases de datos para soportar el análisis, diseño y programación orientada a objetos.

Estos sistemas utilizan las características del lenguaje orientado a objetos para realizar la indexación de la información, con lo cual se vuelve más eficiente y ágil que cualquier otro sistema de administración.

1.3 Características del lenguaje de consulta orientado a objetos

La orientación a objetos afecta no sólo a la especificación del esquema de la base de datos sino también al lenguaje de consulta. Si nos centramos en los lenguajes de consulta orientados a objetos, conviene resaltar tres características que vienen directamente impuestas por el modelo de objetos y que marcarán claramente las diferencias con los lenguajes de consulta relacionales.

- Jerarquías de herencia
- Jerarquías de agregación
- Método

1.3.1 Jerarquías de herencia

La herencia provoca que una instancia de una clase sea también una instancia de su superclase. Y esto implica que cuando se desea realizar una consulta sobre una clase en general incluye a todas sus subclases, a menos que al momento de realizar la consulta se le indique lo contrario.

1.3.2 Jerarquías de agregación

Mientras que en el modelo relacional los valores de los atributos se restringen a tipos primitivos simples, en el modelo orientado a objetos el valor del atributo de un objeto puede ser un objeto o un conjunto de objetos. Esto provoca que las condiciones de búsqueda en una consulta sobre una clase, se puedan seguir expresando de la forma <atributo operador valor>, al igual que en el modelo relacional, pero con la diferencia básica de que el atributo puede ser un atributo anidado de la clase.

1.3.3 Métodos

En el modelo de objetos los métodos definen el comportamiento de los objetos, y al igual que en el predicado de una consulta puede aparecer un atributo, también puede aparecer la invocación de un método y realizar una operación lógica o de comparación con el resultado de dicho método.

1.4 Clasificación de las técnicas de indexación

Las características que vienen directamente impuestas por el modelo de objetos mencionados en la parte anterior, exigen técnicas de indexación que nos permitan un procesamiento más eficiente al momento de la realización de las consultas bajo estas condiciones.

En la actualidad ya son muchas las técnicas de indexación en orientación a objetos, pero estas técnicas tienen bien marcadas sus diferencias por lo que se pueden clasificar en:

- 1 Estructurales
- 2 De comportamiento

1.4.1 Estructurales

Se basan en los atributos de los objetos. Estas técnicas son muy importantes porque la mayoría de los lenguajes de consulta orientados a objetos permiten consultar mediante predicados basados en atributos de objetos. A su vez se puede clasificar en:

- 3 Técnicas basadas en jerarquía de herencia
- 4 Técnicas basadas en jerarquía de agregación
- 5 Técnicas basadas en jerarquía de herencia y de agregación

1.4.1.1 Técnicas basadas en jerarquía de herencia

Estas técnicas se basan en la idea de que una instancia de una subclase es también una instancia de la superclase. Como resultado, el ámbito de acceso de una consulta contra una clase generalmente incluye, no sólo sus instancias sino también las de todas sus subclases.

Con el fin de soportar las relaciones de subclase-superclase eficientemente, el índice debe cumplir dos objetivos: la recuperación eficiente de instancias de una clase simple, y la recuperación eficiente de instancias de clases en una jerarquía de clases.

1.4.1.2 Técnicas basadas en jerarquía de agregación

Estas técnicas se basan en la idea de que las consultas en orientación a objetos soportan predicados anidados. Para soportar dichos predicados, los lenguajes de consulta generalmente proporcionan alguna forma de expresiones de camino.

1.4.1.3 Técnicas basadas en jerarquía de herencia y de agregación

Estas técnicas de indexación proporcionan soporte integrado para consultas que implican atributos anidados de objetos y jerarquías de herencia. Es decir, son técnicas que pretenden mejorar el rendimiento en consultas que contienen un predicado sobre un atributo anidado e implican varias clases en una jerarquía de herencia dada, para que la búsqueda se realice en un único índice.

El grado de complejidad de estas técnicas aumenta considerablemente al ser principalmente combinaciones de las anteriores.

1.4.2 De comportamiento

Proporcionan una ejecución eficiente para consultas que contienen invocación de métodos. La materialización de métodos (*method materialization*) es una de dichas técnicas. En este campo no existe una proliferación de técnicas tan grande como en los anteriores.

El uso de métodos es importante porque define el comportamiento de los objetos, sin embargo, tiene dos efectos negativos desde el punto de vista del procesamiento y optimización de consultas:

- 1 Es caro invocar un método, ya que eso ocasiona la ejecución de un programa, y si esto tiene que realizarse sobre un conjunto grande de objetos degradará el procesamiento de la consulta.
- 2 La implementación de cada método está oculta (por el encapsulamiento del modelo orientado a objetos), por lo que es muy difícil estimar el coste de su invocación en una consulta.

1.4.2.1 Materialización de métodos

La materialización de métodos es una técnica que pretende aliviar los problemas anteriormente mencionados. La idea básica consiste en calcular los resultados de los métodos accedidos frecuentemente, y almacenar los resultados obtenidos para emplearlos más tarde.

2 TÉCNICAS DE INDEXACIÓN

2.1 Técnicas estructurales

2.1.1 Técnicas de jerarquía de herencia

Estas técnicas se basan en la idea de que una instancia de una subclase es también una instancia de la superclase. Como resultado, el ámbito de acceso de una consulta contra una clase generalmente incluye, no sólo sus instancias sino también las de todas sus subclases. Con el fin de soportar las relaciones de subclase-superclase eficientemente, el índice debe cumplir dos objetivos: la recuperación eficiente de instancias de una clase simple, y la recuperación eficiente de instancias de clases en una jerarquía de clases.

2.1.1.1 *Single Class (SC)*

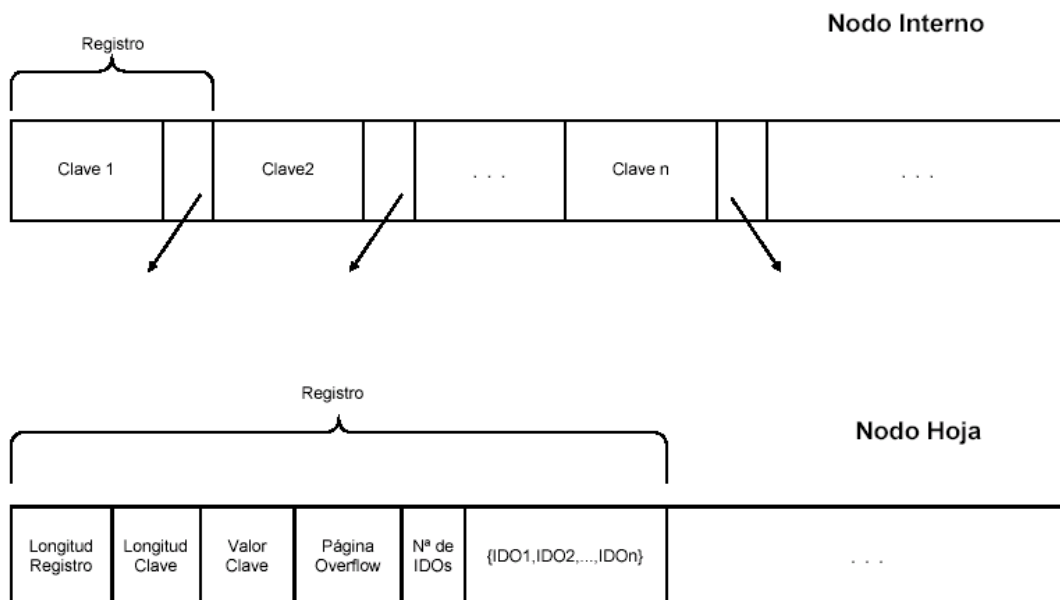
La técnica *Single Class* fue una de las primeras en emplearse. Se basa en la idea tradicional del modelo relacional de un índice para cada relación (en este caso clase) y atributo indexado.

Esto quiere decir que para soportar la evaluación de una consulta cuyo ámbito es una jerarquía de clases, el sistema debe mantener un índice sobre el atributo para cada clase en la jerarquía de clases.

2.1.1.1.1 Estructura

La estructura de esta técnica de indexación se basa en los árboles B⁺. En esta técnica, la creación de un índice para un atributo de un objeto requiere la construcción de un árbol B⁺ para cada clase en la jerarquía indexada como se muestra en la figura 1.

Figura 1. Nodo interno y nodo hoja de un árbol B⁺



2.1.1.1.2 Operaciones

En las operaciones de búsqueda y actualización de los datos del atributo indexados siguen los mismos criterios que los árboles B^+ típicos.

2.1.1.2 CH-Tree

También llamado Árbol de Jerarquía de Clases (*Class Hierarchy Tree - CH Tree*) que se basa en mantener un único árbol, índice para todas las clases de una jerarquía de clases. El *CH-Tree* indexa una jerarquía de clases sobre un atributo común, típicamente uno de los atributos de la superclase, sobre una estructura de un árbol B^+ .

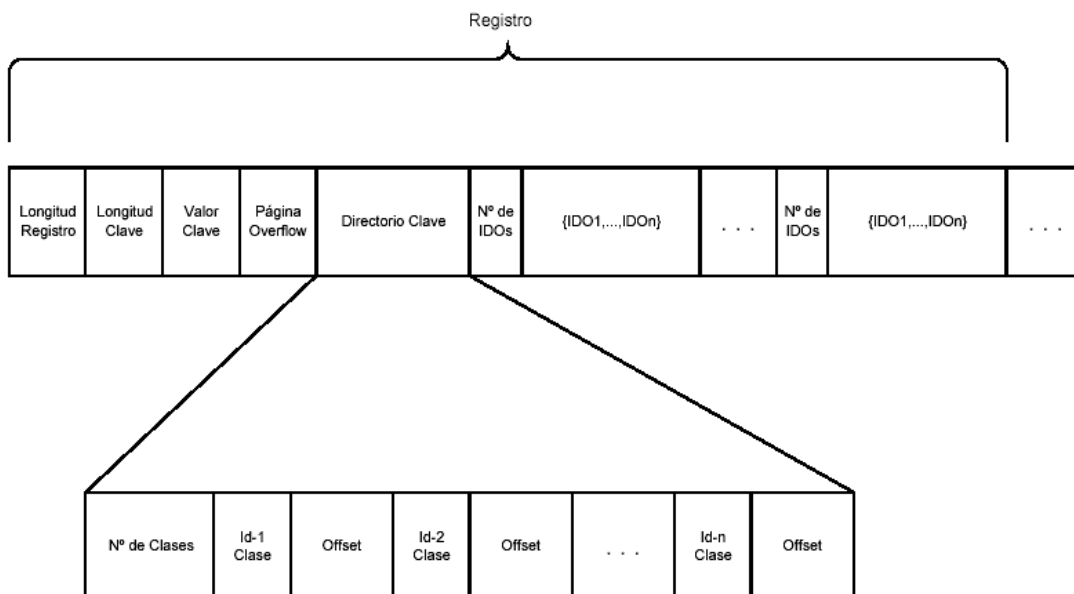
Esta técnica permitirá evaluar consultas realizadas contra una clase simple en la jerarquía de clases, así como contra cualquier sub-jerarquía de dicha jerarquía.

2.1.1.2.1 Estructura

La estructura del *CH-Tree* está basada en los árboles B^+ , y de hecho, el nodo interno es similar al de éstos. La diferencia está en los nodos hoja. Éstos contienen:

- Para cada clase en la jerarquía, el número de elementos almacenados en la lista de identificadores de objetos (IDOs) que corresponden a los objetos que contienen el valor-clave en el atributo indexado, y la propia lista de IDOs.
- Un directorio clave que almacena el número de clases que contienen objetos con el valor-clave en el atributo indexado, y para cada clase, el identificador de la misma y el desplazamiento en el registro índice, que indica donde encontrar la lista de IDOs de los objetos.

Figura 2. Nodo hoja de un CH-Tree



Es decir, el nodo hoja agrupa la lista de IDOs para un valor-clave en términos de las clases a las que pertenecen.

2.1.1.2.2 Operaciones

La búsqueda en un *CH-Tree* es similar a la de los árboles B^+ , de tal manera que, cuando el nodo hoja es localizado, se comparan los valores clave almacenados en el registro índice con la clave buscada. Si coinciden los valores, entonces, si en la consulta se referencia la jerarquía de clases completa, se devuelven todos los IDOs que aparecen en el registro índice. Por el contrario, si en la consulta se referencia sólo una clase, se consulta el directorio clave del registro índice para localizar los IDOs asociados con la clase.

2.1.1.3 H-Tree

Representan una alternativa al *CH-Tree*. Se basan también en los árboles B^+ , y pueden verse como los sucesores de los índices SC. Esta técnica se basa en mantener un *H-Tree* (*Hierarchy Tree*) para cada clase de una jerarquía de clases, y estos *H-Trees* se anidan de acuerdo a sus relaciones de superclase-subclase. Cuando se indexa un atributo, el *H-Tree* de la clase raíz de una jerarquía de clases se anida con los *H-Trees* de todas sus subclases inmediatas, y los *H-Trees* de sus subclases se anidan con los *H-Trees* de sus respectivas subclases y así sucesivamente. Indexando de esta manera se forma una jerarquía de árboles de índices.

2.1.1.3.1 Estructura

La estructura del *H-Tree* se basa en los árboles B^+ , pero presenta variaciones tanto en el nodo interno como en el nodo hoja:

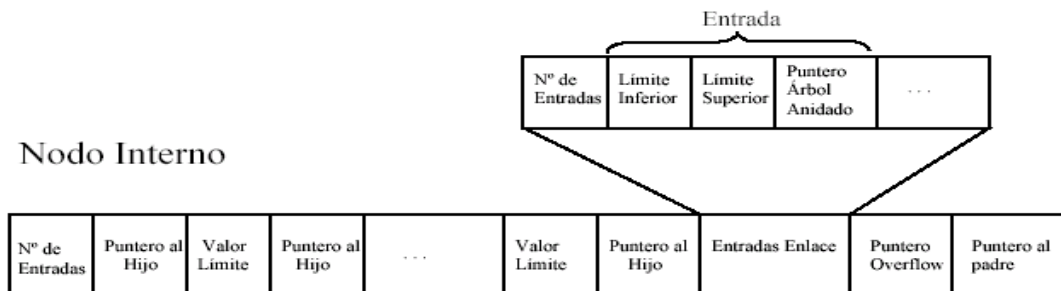
- El nodo hoja contiene, un contador que indica el número de IDOs que hay en la lista de identificadores de objetos cuyo valor en el atributo indexado es el valor-clave, y la propia lista de IDOs.
- El nodo interno aparte de los valores clave discriminantes y los punteros a los nodos hijos, almacena punteros que apuntan a subárboles de *H-Trees* anidados. También almacena el puntero al subárbol anidado, así como los valores máximos y mínimos de dicho subárbol con el fin de evitar recorridos innecesarios del mismo.

Figura 3. Nodo interno y nodo hoja de un *H-Tree*

Nodo Hoja



Nodo Interno



2.1.1.3.2 Operaciones

El anidamiento de los *H-Trees* evita la búsqueda en cada *H-Tree* cuando se consultan un número de clases de la jerarquía. Cuando se busca en los árboles de una clase y sus subclases, se hace una búsqueda completa en el *H-Tree* de la superclase, y una búsqueda parcial en los *H-Trees* de las subclases. Esta es la principal ventaja con relación a los índices SC. Además, un *H-Tree* puede ser accedido independientemente del *H-Tree* de su superclase, ya que la clase consultada no tiene por que ser la clase raíz de la jerarquía de clases, y por tanto, buscar instancias dentro de una sub-jerarquía de clases, puede comenzar en cualquier clase siempre que esté indexada por el mismo atributo. El mayor inconveniente de esta estructura es la dificultad para dinamizarla.

2.1.1.4 hcC-Tree

Los *hcC-trees* (*hierarchy class Chain tree*) representan una alternativa a los esquemas vistos anteriormente, y al igual que ellos permiten la recuperación eficiente tanto de instancias de una clase simple como de una jerarquía de clases, pero para ello almacena información en dos cadenas de cadenas: cadenas de jerarquía (*hierarchy chain*) y cadenas de clase (*class chain*), y al igual que el *CH-Tree* únicamente emplea un árbol para indexar una jerarquía de clases por un atributo. Este esquema pretende la creación de una estructura que favorezca los siguientes tipos de consulta:

- *CHP (Class Hierarchy Point)*: consultas puntuales sobre todas las clases

de una jerarquía de clases que comienza con la clase consultada.

- *SCP (Single Class Point)*: consultas puntuales sobre una clase simple.
- *CHR (Class Hierarchy Range)*: consultas de rango sobre todas las clases

de una jerarquía de clases que comienza con la clase consultada.

- *SCR (Single Class Range)*: consultas de rango sobre una clase simple.

Atendiendo a esta clasificación, se observa que tanto las consultas *CHP* como las *CHR* se favorecerían si los identificadores de los objetos estuvieran agrupados, para todas las clases de la jerarquía con un valor dado en el atributo indexado. Y por otro lado, las consultas *SCP* y *SCR* se favorecerían si los identificadores de los objetos de una clase simple para un valor concreto del atributo indexado estuvieran agrupados juntos. A priori, ambos requerimientos

como se aprecia entran en conflicto, y para evitarlo se diseña esta estructura que da solución a ambos.

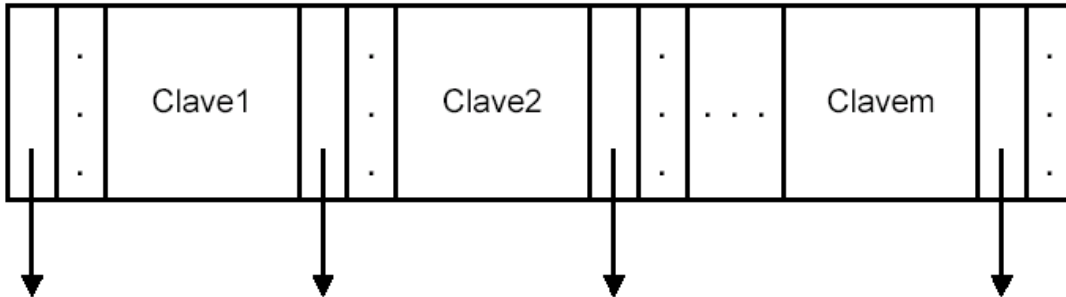
2.1.1.4.1 Estructura

La estructura del *hcC-Tree* se basa en los árboles B^+ , pero a diferencia de éstos distingue tres tipos de nodos:

- **Nodos internos.** Constituyen los niveles superiores del árbol. La estructura del nodo interno está constituida por m claves y $m+1$ punteros (definición similar al nodo interno del árbol B^+), pero además, para cada uno de los $m+1$ intervalos se almacena un *bitmap* de n *bits*.

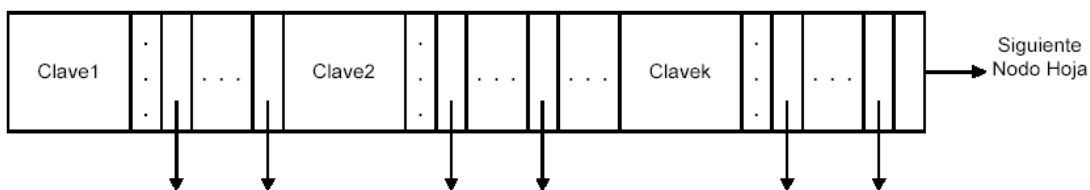
Un *bit* dentro del *bitmap* correspondiente a una clase para un intervalo, es 0 si y sólo si no existen objetos pertenecientes a esa clase que tengan un valor para el atributo indexado que esté dentro del intervalo.

Figura 4. Nodo interno del *hcC-Tree*



- **Nodos hoja.** Representan los penúltimos niveles del árbol. Cada entrada en el nodo hoja está constituido por un valor clave (k), un *bitmap* (de n bits, uno por clase) y un conjunto de punteros. Un *bit* correspondiente a una clase es asignado si y solo si existe al menos un objeto perteneciente a esa clase que tiene el valor clave k . Si suponemos que son asignados m bits en el *bitmap* para el valor k , entonces el conjunto de punteros consta de $m+1$ punteros, m de ellos apuntan a nodos-IDO de cadenas de clase y uno a un nodo-IDO de cadenas de jerarquía.

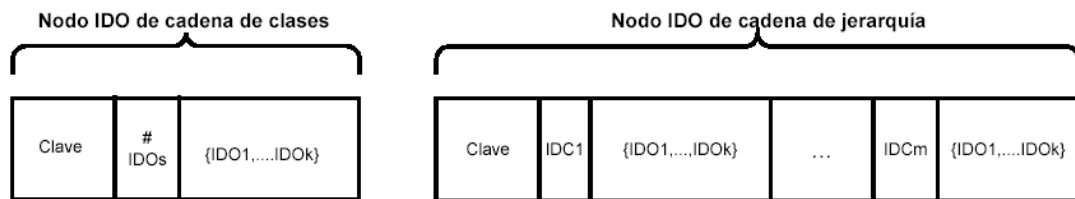
Figura 5. Nodo hoja de un *hcC-Tree*



- **Nodo IDO.** Representan el último nivel del árbol. Para cada clase en la jerarquía hay una cadena de nodos-IDO (cadena de clase), y hay sólo una cadena de nodos-IDO que corresponden a todas las clases (cadena de jerarquía). Los nodos-IDO de una cadena de

clase almacenan el IDO de los objetos de esa clase solamente (cada entrada es de la forma <clave, lista IDOs>). Los nodos-IDO de una cadena de jerarquía contienen los IDO de los objetos pertenecientes a todas las clases (una entrada es de la forma <clave, conjunto de lista de IDOs>). Los nodos-IDO de cada una de las n+1 cadenas son enlazados para facilitar el recorrido de los nodos de izquierda a derecha.

Figura 6. Entrada de nodos-IDO del *hcC-Tree*



2.1.1.4.2 Operaciones

La operación de búsqueda en un *hcC-Tree* es similar a la de los B^+ , pero teniendo en cuenta que en función del tipo de consulta (SCP, SCR, CHR o CHP) se examinará o bien la cadena de jerarquía o bien la cadena de clase, y además, se utilizarán los *bit* asignados en los *bitmap* para evitar recorridos innecesarios.

El algoritmo de inserción se divide en tres partes: la primera es similar al algoritmo de inserción de los B⁺, la segunda consiste en añadir el IDO a la cadena de clase, y la tercera consiste en añadir el IDO a la cadena de jerarquía. Al analizar estas operaciones se observa que pueden ser necesarios como máximo tres recorridos extra para la inserción, pero en la mayoría de los casos no habrá sobrecarga extra. Por lo tanto, en el peor de los casos el costo del algoritmo de inserción está limitado por la altura del árbol.

2.1.2 Técnicas de jerarquía de agregación

Estas técnicas se basan en la idea de que las consultas en orientación a objetos soportan predicados anidados. Para soportar dichos predicados, los lenguajes de consulta generalmente proporcionan alguna forma de expresiones de camino. Previo al análisis de estas técnicas de indexación es necesario concretar el significado de algunos conceptos.

Camino. Un camino es una rama en una jerarquía de agregación que comienza con una clase C y termina con un atributo anidado de C. Formalmente, un camino P para una jerarquía de clases H se define como C(1).A(1).A(2)...A(n), donde:

- C(1) es el nombre de una clase en H,
- A(1) es un atributo de la clase C(1), y

- $A(i)$ es un atributo de una clase $C(i)$ en H , tal que $C(i)$ es el dominio del atributo $A(i-1)$ de la clase $C(i-1)$, $1 < i \leq n$.

Instancia de un camino. La instancia de un camino representa la secuencia de objetos obtenida por la instancia de las clases que pertenecen al camino. Formalmente, dado un camino $P = C(1).A(1).A(2)...A(n)$, una instancia de P se define como una secuencia de $n+1$ objetos, denotados como $O(1).O(2)...O(n+1)$, donde:

1. $O(1)$ es una instancia de la clase $C(1)$, y
2. $O(i)$ es el valor de un atributo $A(i-1)$ del objeto $O(i-1)$, $1 < i \leq n+1$.

Instancia parcial. Una instancia parcial es que comienza con un objeto que no es una instancia de la primera clase a lo largo del camino, sino de alguna otra clase del camino. Formalmente, dado un camino $P = C(1).A(1).A(2)...A(n)$, una instancia parcial de P es definida como una secuencia de objetos $O(1).O(2)...O(j)$, $j < n+1$, donde:

- 3 $O(1)$ es una instancia de una clase $C(k)$ en $Class(P)$ para $k=n-j+2$,
- 4 $O(i)$ es el valor del atributo $A(i-1)$ del objeto $O(i-1)$, $1 < i \leq j$, y siendo
- 5 $Class(P) = C(1) \cup \{C(i) \text{ tal que } C(i) \text{ es el dominio del atributo } A(i-1) \text{ de la clase } C(i-1), 1 < i \leq n\}$.

Instancia parcial no redundante. Una instancia de camino es no redundante si no está contenida en otra instancia más larga. Formalmente, dada una instancia parcial $p = O(1).O(2)...O(j)$ de P , p es no redundante, si no existe una instancia $p' = O'(1).O'(2)...O'(k)$, $k > j$, tal que $O(i) = O'(k-j+i)$, $1 \leq i \leq j$.

5.1.1.1 *Path Index (PX)*

Un *path index* (índice de camino), es un índice que almacena la instancia de un camino, la cual es una secuencia de objetos, siendo la clave del índice el objeto al final del camino de instancia. Formalmente, dado un camino $P = C(1).A(1).A(2)...A(n)$, un índice de camino (PX) sobre P es definido como un conjunto de pares (O,S) , donde

2. O es el valor clave, y

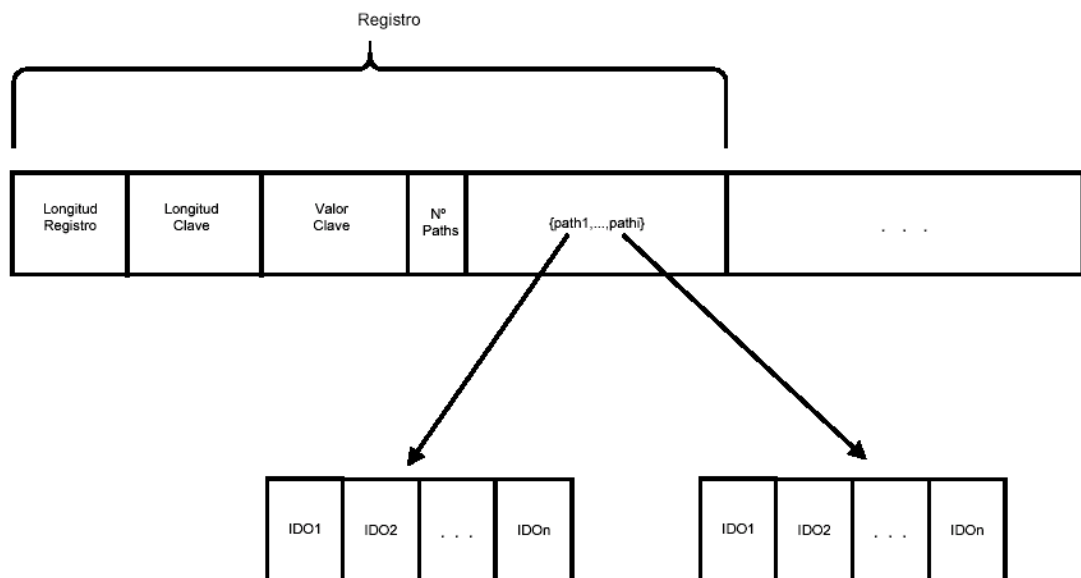
- $S = \{\pi_{\langle j-1 \rangle}(p(i)) \mid p(i) = O(1).O(2)...O(j) \text{ (} 1 \leq j \leq n+1 \text{)} \}$ es una instancia no redundante (parcial o no) de P , y $O(j) = O$. $\pi_{\langle j-1 \rangle}(p(i))$ denota la proyección de $p(i)$ sobre los primeros $j-1$ elementos.

De esto se deduce que, un índice PX almacena todos los objetos que finalizan con esa clave.

5.1.1.1.1 Estructura

La estructura de datos base para este índice puede ser un árbol B⁺, al igual que para los *Nested* y *Multiindex* que se verán a continuación, y de hecho, el formato de los nodos internos es idéntico en los tres casos. La variación está en el nodo hoja.

Figura 7. Nodo hoja para un índice PX



En el caso del índice PX, el nodo hoja (Figura 7) contiene entre otra información, el número de elementos en la lista de instancias que tienen como objeto final el valor clave, y la propia lista de instancias. Cada instancia de camino se implementa por un arreglo que tiene la misma longitud que el camino. El primer elemento del arreglo es el IDO de una instancia de la clase C(1), el segundo elemento es el IDO de la instancia de la clase C(2) referida por el atributo A(1) de la instancia del primer IDO del arreglo, y así sucesivamente.

5.1.1.1.2 Operaciones

En un índice PX para evaluar un predicado sobre un atributo $A(n)$ de la clase $C(i)$ ($1 \leq i \leq n$) del camino P , sólo se necesita recorrer un índice. Una vez que se han determinado las instancias de camino asociadas con el valor de la clave, se extraen los IDOs que ocupan la posición i -ésima de cada arreglo.

Las actualizaciones sobre un PX son costosas, ya que se requieren recorridos hacia delante, sin embargo no es necesario un recorrido inverso, ya que en los nodos hoja se almacenan las instancias completas del camino. Esta organización se puede emplear aunque los objetos no contengan referencias inversas al no ser necesario el recorrido inverso.

5.1.1.2 *Nested* (NX)

Un *Nested Index* (Índice Anidado) establece una conexión directa entre el objeto al comienzo y el objeto al final de un camino de instancia, siendo la clave del índice el objeto al final de dicho camino. Formalmente, dado un camino $P = C(1).A(1).A(2)...A(n)$, un índice (NX) sobre P se define como un conjunto de pares (O,S) , donde:

- O es el valor clave, y
- S es el conjunto de IDOs de $C(1)$, tales que O y cada IDO en S, aparecen en la misma instancia del camino.

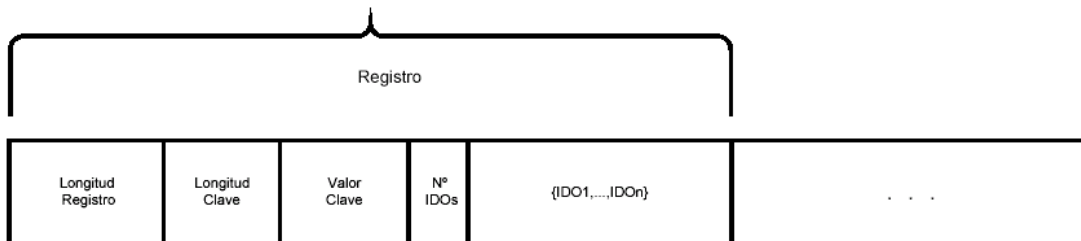
Un índice NX a diferencia del índice PX sólo almacena los objetos iniciales de las instancias de los caminos. Como se puede observar cuando $n=1$, el NX y el PX son idénticos, y son los índices empleados en la mayoría de los sistemas de bases de datos relacionales.

5.1.1.2.1 Estructura

El nodo interno es similar al de los árboles B^+ típicos. El nodo hoja para el NX simplemente almacena la longitud del registro, la longitud de la clave, el valor clave, el número de elementos en la lista de IDOs de los objetos que contienen el valor clave en el atributo indexado, y la lista de IDOs.

La lista de IDOs está ordenada, por lo que cuando el tamaño de un registro excede el tamaño de la página, se mantiene un pequeño directorio al comienzo del registro. Este directorio contiene la dirección de cada página del registro y el valor mayor contenido en la página. Cuando se añade o se elimina un IDO de un registro se puede identificar de manera directa la página en la que hay que hacer la modificación.

Figura 8. Nodo hoja de un índice *Nested*



5.1.1.2.2 Operaciones

La actualización con estos índices si es problemática, ya que requieren el acceso a varios objetos, con vistas a determinar que entradas de índice se deben actualizar. Es necesario tanto del recorrido hacia delante como del recorrido inverso de los objetos. El recorrido hacia delante se necesita para determinar el valor del atributo indexado (valor del atributo final del camino) del objeto modificado. El recorrido inverso es necesario para determinar todas las instancias al comienzo del camino, y es muy costoso cuando no hay referencias inversas entre objetos.

5.1.1.3 Multiindex (MX)

Esta organización se basa en asignar un índice a cada una de las clases que constituyen el camino.

Formalmente, dado un camino $P = C(1).A(1).A(2)...A(n)$, un *Multiindex* (MX) se define como un conjunto de n índices *Nested* (NX), $NX.1, NX.2, \dots NX.n$, donde $NX.i$ es un índice *Nested* definido sobre el camino $C(i).A(i)$, $1 \leq i \leq n$. Esta técnica es empleada en el sistema *GemStone*.

5.1.1.3.1 Estructura

La estructura de estos índices se basa en un árbol B^+ al igual que los anteriores. Y tanto su nodo interno como sus nodos hoja tienen una estructura similar a la de los *Nested* (véase Figura 8)

5.1.1.3.2 Operaciones

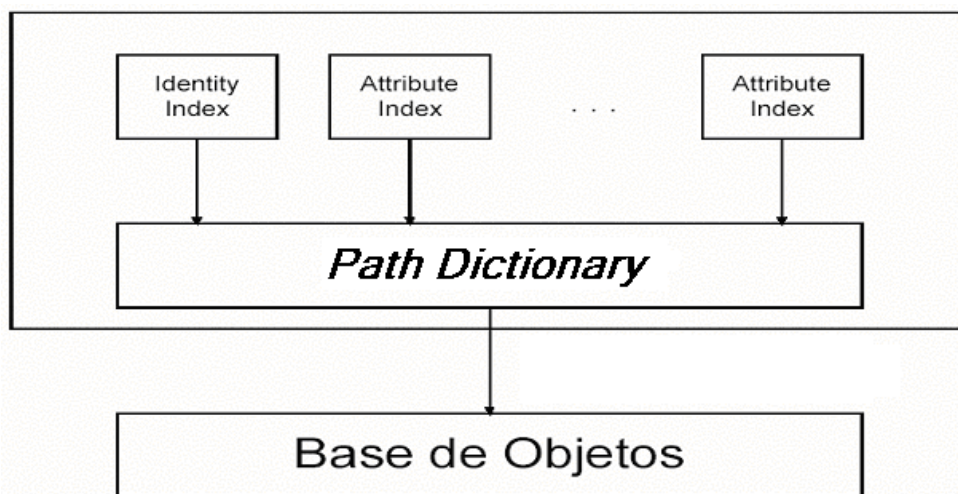
Con esta organización la operación de recuperación se lleva a cabo recorriendo primero el último índice asignado sobre el camino. El resultado de este recorrido se usa como clave para la búsqueda sobre el índice que precede a este último, y así sucesivamente hasta que se recorra el primer índice.

La mayor ventaja de esta técnica es el bajo costo de actualización, sin embargo, el resolver un predicado anidado requiere recorrer una cantidad de índices igual a la longitud del camino.

5.1.1.4 *Path Dictionary Index (PDI)*

Un *Path Dictionary Index* (Índice Diccionario de Caminos) es una estructura de acceso que reduce los costos de procesamiento de las consultas que soportan tanto búsquedas asociativas como recorridos de objetos. Para ello, consta de dos partes (Figura 9): el *path dictionary*, que soporta un recorrido eficiente de los objetos, y los índices *identity* y *attribute* que soportan una búsqueda asociativa, y son construidos sobre el *path dictionary*.

Figura 9. Índice diccionario de caminos



Path Dictionary (PD). El *path dictionary* extrae los atributos complejos de la base de datos para representar las conexiones entre objetos. Evita por tanto, un acceso innecesario a los objetos en la base de datos almacenando la información del camino entre los objetos en una estructura de acceso separada. Y, ya que los valores de los atributos primitivos no son almacenados, es mucho

más rápido recorrer los nodos en el *path dictionary* que los objetos en la base de datos.

Attribute Index (AI). El *path dictionary* soporta un recorrido rápido entre objetos, pero no ayuda en la evaluación de predicados que implican la búsqueda de objetos que cumplen ciertas condiciones especificadas sobre sus atributos. Para facilitar esta búsqueda el PDI proporciona los índices *attribute*, que asocian los valores de los atributos a los IDOs en el *path dictionary* que corresponden a esos valores. En lugar de traducir valores de atributos a objetos directamente (como *Nested* o *Path*), el índice *attribute* traduce valores de atributos a información del camino almacenado en el *path dictionary*.

El *path dictionary* sirve como una estructura compartida para el recorrido de objetos, y como un nivel de separación desde los valores de los atributos a los objetos físicos.

Esta separación entre el soporte para el recorrido y para las búsquedas asociativas permite construir tantos índices *attribute* sobre el *path dictionary* como sea necesario sin incurrir en un crecimiento extraordinario en la sobrecarga de almacenamiento. Esta separación se traduce también en un bajo costo de mantenimiento.

Identity Index (II). Como los IDOs son empleados para describir la información del camino entre objetos, a menudo se necesita obtener información en el *path dictionary* sobre el camino asociado con un determinado IDO. Con el fin de soportar eficientemente esta posibilidad se crea un índice *identity* que proporciona una asociación de los IDOs con las localizaciones en el *path*

dictionary donde pueden encontrarse dichos IDOs. Este índice *identity* reduce significativamente el costo de las operaciones de recuperación y actualización, y al igual que ocurre con el índice *attribute*, este índice es organizado como un árbol de búsqueda separado sobre el *path dictionary*.

5.1.1.4.1 Estructura

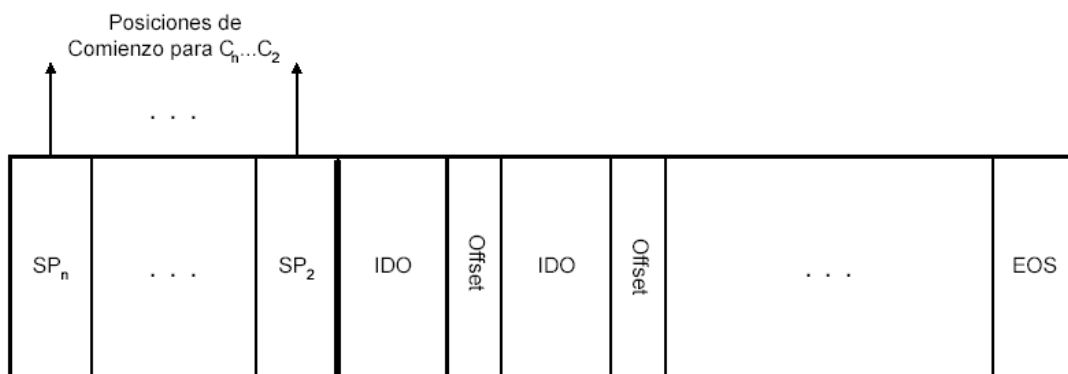
Esta técnica de indexación como se ha visto con anterioridad consta de diversas estructuras, una para cada tipo de índice empleado. A continuación se describen brevemente las estructuras de cada uno de ellos.

Path Dictionary. Una implementación del *path dictionary* es el esquema s-expresión, donde cada s-expresión en el *path dictionary* almacena la información del camino para un subárbol de objetos; es decir, codifica en una expresión todos los caminos que terminan con el mismo objeto en una clase hoja. La s-expresión para un camino $C(1).C(2).C(n)$ se define como

- $S(1) = \emptyset_1$, donde \emptyset_1 es el IDO de un objeto en la clase $C(1)$ o nulo.
- $S(i) = \emptyset_i(S(i-1)[,S(i-1)])$ $1 < i \leq n$, donde \emptyset_i es el IDO de un objeto en la clase $C(i)$ o nulo, y $S(i-1)$ es una s-expresión para el camino $C(1)C(2)...C(i-1)$

El *path dictionary* para $C(1).C(2).C(n)$ consta de una secuencia de n -niveles de *s*-expresiones. El objeto que va en cabeza en una *s*-expresión, que no tiene porque pertenecer necesariamente a $C(n)$ es el objeto terminal de los caminos denotados por la *s*-expresión. Así, el número de *s*-expresiones que corresponden a un camino es igual al número de objetos a lo largo del camino que no tiene un objeto anidado sobre el camino.

Figura 10. Estructura de datos de una *s*-expresión

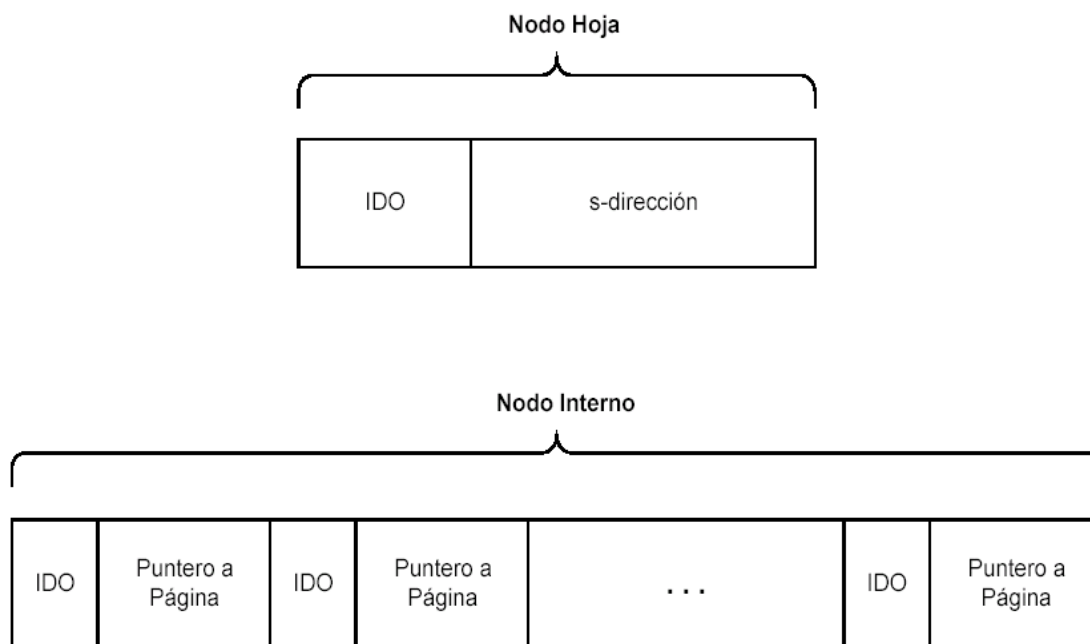


Los punteros SP_i apuntan a la primera ocurrencia de \emptyset_i en la expresión. Siguiendo los campos SP_i hay una serie de pares $\langle \text{IDO}, \text{Offset} \rangle$, donde los *Offsets* asociados con un \emptyset_i ($2 \leq i \leq n$) apuntan a la siguiente ocurrencia de \emptyset_i en la *s*-expresión. Al final de la *s*-expresión hay un símbolo especial de *end-of-s-expresión* (EOS).

Identity Index. Su estructura está basada también en los árboles B^+ , pero tanto el nodo interno como el nodo hoja contienen algunas variaciones. En este índice los IDOs son empleados como valor clave (Figura 11). La *s*-dirección en el nodo hoja es la dirección de la *s*-expresión que corresponde al IDO que está

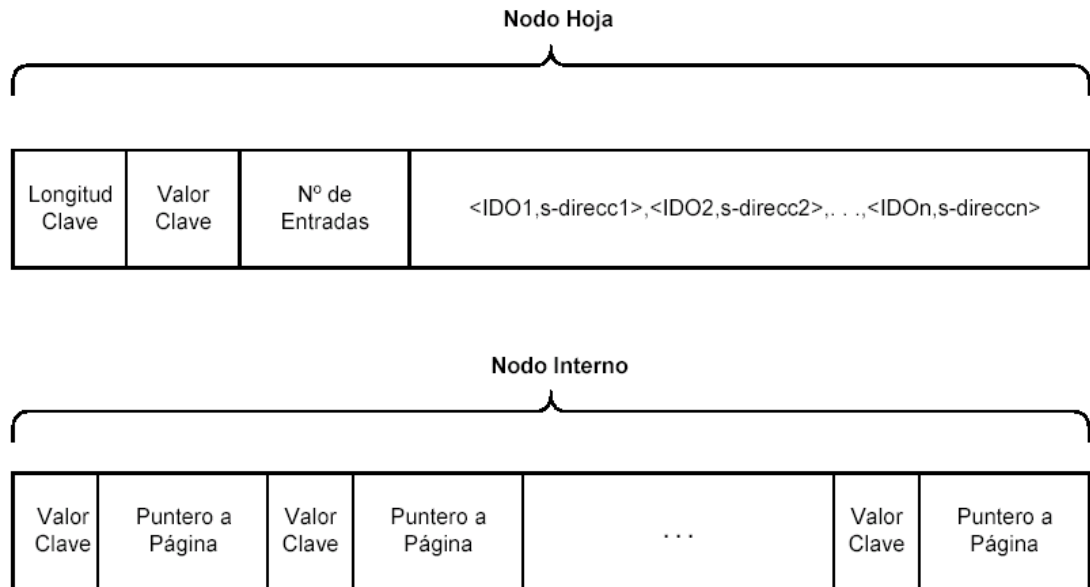
en ese nodo hoja. Los punteros de página en un nodo interno están apuntando al siguiente nivel de nodos internos o a los nodos hoja.

Figura 11. Nodo hoja e interno de un índice *identity*



Attribute Index. La estructura de este índice está basada también en los árboles B^+ . Los IDOs y las direcciones de las s-expresiones (s-direcc) (Figura 12) son empleados para acceder a las s-expresiones de los correspondientes IDOs.

Figura 12. Nodos hoja e interno de un índice *attribute*



5.1.1.4.2 Operaciones

Los índices *attribute* proporcionan ventajas para el procesamiento de predicados simples con operaciones de rango e igualdad. Así, por ejemplo, en una operación de igualdad se emplea el valor del atributo especificado en el predicado para buscar en el índice *attribute* las correspondientes direcciones de las s-expresiones, y a través de estas direcciones se pueden obtener las s-expresiones y derivar desde las s-expresiones los IDOs de la clase.

Si la operación es de desigualdad, el índice *attribute* no puede emplearse, pero sí el índice *identity* y el *path dictionary*. Los índices *attribute*

tampoco proporcionan beneficios para la evaluación de predicados complejos; estos requieren la búsqueda también en el índice *identity* o buscar secuencialmente en el *path dictionary*.

5.1.1.5 *Join Index Hierarchy* (JIH)

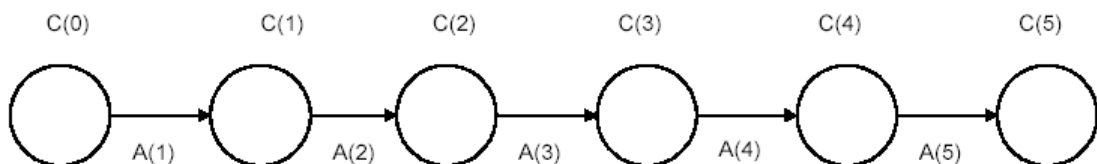
La técnica *Join Index Hierarchy* (Jerarquía de Índices *Join*), se basa en extender el índice *join* de las bases de datos relacionales y, sus variaciones en bases de datos espaciales para construir jerarquías de índices *join* que aceleren la navegación entre secuencias de objetos y clases.

En esta técnica, un índice *join* almacena pares de identificadores de objetos de dos clases que están conectadas por medio de relaciones lógicas directas (índices *join* base) o indirectas (índices *join* derivados). JIH soporta el recorrido hacia delante y hacia atrás en una secuencia de objetos y clases, y además, soporta una propagación de actualizaciones eficiente comenzando en los índices *join* base y localizando la propagación de actualizaciones en la jerarquía. Esta estructura es especialmente apropiada para navegaciones frecuentes y actualizaciones poco frecuentes.

***Schema Path* (Camino de Esquema).** Un esquema de una base de datos es un grafo dirigido en el que los nodos corresponden a clases, y las flechas a

relaciones entre clases. Así, si $A(k)$ es un atributo de la clase $C(i)$, y su rango es la clase $C(j)$, entonces existe una flecha dirigida de $C(i)$ a $C(j)$ etiquetada como $A(k)$. Además, si para $i=0,1,\dots,n-1$ hay una flecha dirigida de $C(i)$ a $C(i+1)$, etiquetada como $A(i+1)$, en el esquema de la base de datos, entonces $\langle C(0),A(1),C(1),A(2),\dots,A(n),C(n) \rangle$ es un camino de esquema (Figura 13).

Figura 13. Camino de esquema de longitud 5



Join Index File (archivo de índice Join). Dado un camino de esquema $\langle C(0),A(1),C(1),A(2),\dots,A(n),C(n) \rangle$ sobre un esquema de una base de datos, un archivo de índice *join*, $Jl(i,j)$ ($1 \leq i < j \leq n$) consta de un conjunto de tuplas $(IDO(o_i), IDO(o_j), m)$, donde o_i y o_j son objetos de clases $C(i)$ y $C(j)$, respectivamente, y existe al menos un camino de objetos $\langle o_i, o_{i+1}, \dots, o_{j-1}, o_j \rangle$ tal que para $k=0,1,\dots,j-i-1$, el objeto o_{i+k+1} es referenciado por o_{i+k} por medio del atributo $A(i+k+1)$, y m es el número de caminos de objetos distintos que conectan los objetos o_i y o_j .

Join Index Hierarchy (jerarquía de índices Join). Una jerarquía de índices *join* $JIH(C(0),A(1),C(1),A(2),\dots,A(n),C(n))$ o $JIH(0,n)$ está constituida por nodos

de índices *join* que conectan diferentes objetos a lo largo de un camino de esquema. $Jl(0,n)$ representa la raíz de la jerarquía, y cada nodo $Jl(i,j)$, donde $j-i > 1$, puede tener dos hijos directos $Jl(i,j-k)$ y $Jl(i+l,j)$ donde $0 < k < j-i$ y $0 < l < j-i$. Esta relación de padre-hijo representa una dependencia para actualización entre el nodo hijo y el padre, de forma que cuando el nodo hijo es actualizado, el padre debería ser actualizado también.

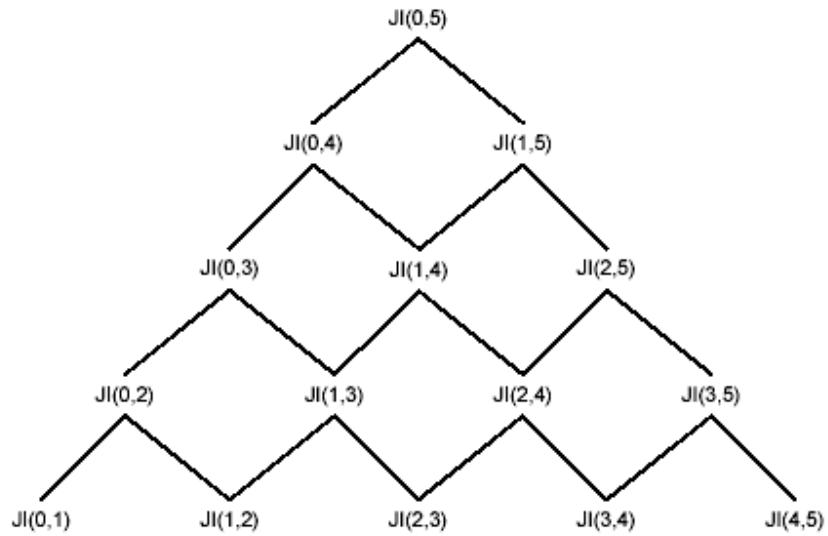
En la parte inferior de la jerarquía están los nodos $Jl(i,i+1)$ para $i=0,1,\dots,n-1$, que constituyen los índices *join* base.

5.1.1.5.1 Estructura

La JIH presenta diferentes variaciones en función de la riqueza de las estructuras de los índices *join* derivados. Así, se presentan tres estructuras para JIH: completa, base y parcial, y en cualquiera de ellas los nodos que constituyen los Jl suelen basarse en árboles B^+ .

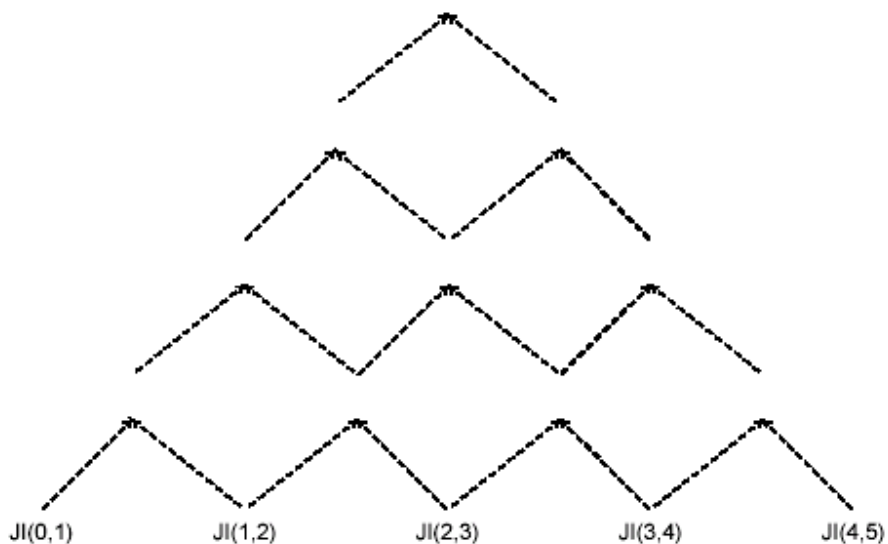
Jerarquía de Índices Join Completa (JIH-C). Está formada por el conjunto completo de todos los posibles índices *join* base y derivados. Soporta, por tanto, la navegación entre dos objetos cualesquiera del camino de esquemas conectados directa o indirectamente.

Figura 14. JIH completa para el camino de esquema de la Figura 13



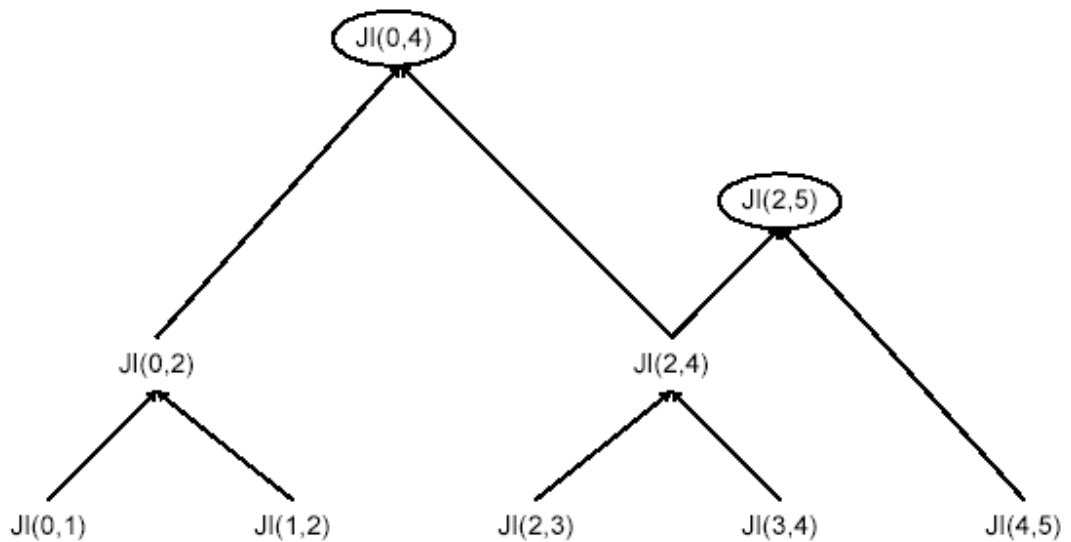
Jerarquía de Índices Join Base (JIH-B). Está formada únicamente por los índices *join* base, por lo que soporta una navegación directa solamente entre dos cualesquiera clases adyacentes. En este caso es una jerarquía degenerada en la que no aparecen los nodos JI de más alto nivel pero que pueden ser derivados desde los índices *join* base.

Figura 15. JIH base para el camino de esquema de la Figura 13



Jerarquía de Índices Join Parcial (JIH-P). Está formada por un subconjunto apropiado de índices *join* base y derivados de una jerarquía de JI completa. Soporta navegaciones directas entre un conjunto de pares de clases pre-especificadas ya que materializa únicamente a los JI correspondientes y sus índices *join* derivados (auxiliares) relacionados.

Figura 16. JIH parcial para el camino de esquema de la Figura 13



5.1.1.5.2 Operaciones

De las tres estructuras anteriores, la parcial es la más sugerente ya que parece más beneficioso materializar únicamente los índices *join* de la jerarquía que soportan las navegaciones empleadas más frecuentemente.

Ante las operaciones de actualización (inserción, borrado y modificación) en los JI base es necesaria una propagación de esas actualizaciones a los JI derivados. La propagación de la actualización se realiza mediante el operador

.c, que es similar al operador *join* en bases de datos relacionales. Así, $Jl(i,k) \cdot_c Jl(k,j)$ contiene una tupla $(IDO(o_i), IDO(o_j), m1_m2)$ si hay una tupla $(IDO(o_i), IDO(o_k), m1)$ en $Jl(i,k)$ y una tupla $(IDO(o_k), IDO(o_j), m2)$ en $Jl(k,j)$. Una actualización sobre un Jl de un nivel bajo $Jl(C(i), C(k))$ es propagado a un nivel Jl de nivel superior, $Jl(C(i), C(j))$, $j > k$, añadiendo $Jl(i,k) \cdot_c Jl(k,j)$ a $Jl(i,j)$.

Las tuplas idénticas se juntan en una única en la que se ha acumulado el número de caminos. En el caso del borrado si el contador es 0, se debe eliminar la tupla correspondiente.

5.1.2 Técnicas de jerarquía de herencia y agregación

Estas técnicas de indexación proporcionan soporte integrado para consultas que implican atributos anidados de objetos y jerarquías de herencia. Es decir, son técnicas que pretenden mejorar el rendimiento en consultas que contienen un predicado sobre un atributo anidado e implican varias clases en una jerarquía de herencia dada.

El grado de complejidad de estas técnicas aumenta considerablemente al ser principalmente combinaciones de las anteriores, por lo que a continuación se describen muy brevemente.

5.1.2.1 *Nested Inherited*

Dado un camino $P = C(1).A(1).A(2)...A(n)$, un índice *Nested Inherited* (NIX) asocia el valor v del atributo $A(n)$ con los IDOs de las instancias de cada clase en el ámbito de P que tienen v como valor del atributo anidado $A(n)$. Entendiéndose por ámbito de un camino, el conjunto de todas las clases y subclases del camino.

5.1.2.1.1 Estructura

El formato de un nodo no hoja tiene una estructura similar a los índices tradicionales basados en árboles B^+ . Sin embargo, la organización del registro de un nodo hoja, llamado también registro primario, se basa en el del *CH-Tree*, en el sentido de que contiene un directorio que asocia a cada clase el desplazamiento, dentro del registro, donde son almacenados los IDOs de las instancias de la clase. Sin embargo, un NIX puede ser empleado para consultar todas las jerarquías de clases encontradas a lo largo de un camino, mientras que un *CH-Tree* sólo permite consultar una única jerarquía de clases.

La organización del índice consta de dos índices. El primero, llamado índice primario, que es indexado sobre los valores del atributo $A(n)$. Asocia con un valor v de $A(n)$, el conjunto de IDOs de las instancias de todas las clases que tienen v como valor del atributo anidado $A(n)$.

El segundo índice, llamado índice auxiliar, tiene IDOs como claves de indexación. Asocia con el IDO de un objeto la lista de IDOs de sus padres. Los registros del nodo hoja en el índice primario contienen punteros a los registros

del nodo hoja del índice auxiliar, y viceversa. El índice primario es utilizado para las operaciones de recuperación, mientras que el índice secundario es empleado, básicamente, para determinar todos los registros primarios donde son almacenados los IDOs de una instancia dada, con el fin de realizar eficientemente las operaciones de inserción y borrado.

5.1.2.1.2 Operaciones

Esta técnica es apropiada para evaluaciones rápidas de predicados sobre el atributo indexado en consultas centradas en una clase, o jerarquía de clases, para el ámbito del camino que finaliza con el atributo indexado. Primeramente, se ejecuta una búsqueda sobre el índice primario con el valor clave igual al buscado. Entonces es accedido el registro primario, y se realiza una búsqueda en el directorio de clases para determinar el desplazamiento dónde están los IDOs de la clase consultada. Si la consulta se realiza no sobre la clase si no sobre toda la jerarquía cuya raíz es la clase consultada se realizan los mismo pasos pero con la diferencia de que la búsqueda en el directorio de clases se realiza para todas las clases de la jerarquía, con lo que hay que acceder a diferentes partes del registro (uno para cada desplazamiento obtenido desde la búsqueda en el directorio de clases).

5.1.2.2 *Inherited MultiIndex*

Dado un camino $P = C(1).A(1).A(2)...A(n)$ un *Inherited MultiIndex* (IMX) se basa en asociar un índice sobre cada clase $C(i)$, del conjunto de clases que interviene en el camino, que asocia los valores del atributo $A(i)$ con los IDOs de las instancias de $C(i)$ y todas sus subclases.

5.1.2.2.1 Estructura

La estructura de esta técnica son los *CH-Trees* que se basa en los árboles B^+ . Se dispondrá de un número de índices igual a la longitud del camino, ya que se empleará un índice *CH-Tree* para cada jerarquía de herencia que exista en el camino que se está indexando.

5.1.2.2.2 Operaciones

Para la recuperación el IMX necesita recorrer un número de índices igual al número de clases del camino, independientemente de que se consulte únicamente la clase raíz de una jerarquía o bien toda la jerarquía.

5.2 Técnicas de comportamiento

5.2.1 Técnicas basadas en la invocación de métodos

El uso de métodos es importante porque define el comportamiento de los objetos, sin embargo, tiene dos efectos negativos desde el punto de vista del procesamiento y optimización de consultas:

- Es caro invocar un método, ya que eso ocasiona la ejecución de un programa, y si esto tiene que realizarse sobre un conjunto grande de objetos disminuirá drásticamente el procesamiento de la consulta.

- La implementación de cada método está oculta por el encapsulamiento del modelo orientado a objetos, por lo que es muy difícil estimar el coste de su invocación en una consulta.

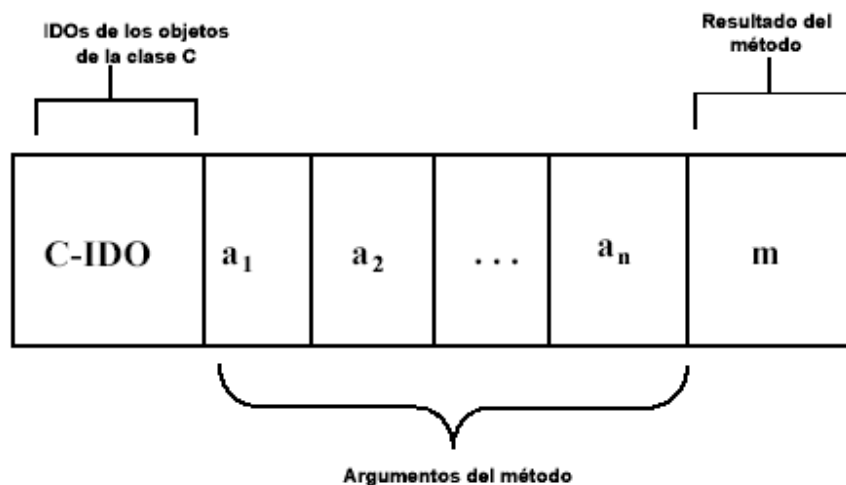
5.2.1.1 Materialización de métodos

La materialización de métodos (MM), es una técnica que pretende aliviar los problemas anteriormente mencionados. La idea básica consiste en calcular los resultados de los métodos accedidos frecuentemente, a priori, y almacenar los resultados obtenidos para emplearlos más tarde.

5.2.1.1.1 Estructura

El resultado materializado de un método puede ser almacenado en una tabla con la estructura mostrada en la figura 17. Como el tamaño de esta tabla puede llegar a ser muy grande, se pueden crear índices para acelerar la velocidad de acceso a la misma.

Figura 17. Formato de la tabla que almacena los métodos materializados



Cuando varios métodos para la misma clase tengan el mismo conjunto de argumentos, los resultados materializados de estos métodos pueden ser almacenados en la misma tabla, con lo que se necesita menos espacio, y si además estos métodos son referenciados en la misma consulta, sólo sería necesario acceder a una tabla para evaluar estos métodos, lo que se traduciría en un coste de procesamiento más bajo.

Si los resultados de cada método referenciado en una consulta han sido precalculados, será mucho más fácil para el optimizador de consultas generar un buen plan de ejecución para la consulta.

5.2.1.1.2 Operaciones

La materialización de métodos permite un procesamiento más eficiente de las consultas de recuperación, pero supone un costo más alto para las consultas de actualización, ya que, una operación de actualización puede ocasionar que el resultado precalculado llegue a ser inconsistente con los datos actualizados.

Una cuestión importante a tener en cuenta en la materialización de métodos es el manejo eficiente de este problema de inconsistencia. Así, cuando se borra un objeto, todas las entradas materializadas que usan ese objeto deben ser borradas, y cuando un objeto existente es modificado, todos los resultados materializados que usen el objeto modificado necesitan ser recalculados. Este proceso puede ser inmediato, o retrasado hasta que el resultado de la instancia del método sea necesario para evaluar la consulta (*lazy materialization*).

6 COMPARACIONES DE EFICIENCIA ENTRE LAS TÉCNICAS DE INDEXACIÓN

6.1 Técnicas estructurales

6.1.1 Técnicas de jerarquía de herencia

6.1.1.1 *Single Class (SC)*

6.1.1.1.1 Adecuado para consultas en una única clase de la jerarquía

Esta técnica de indexación parece apropiada cuando las búsquedas se realizan en una clase concreta dentro de una jerarquía, ya que esto exige únicamente el recorrido del árbol correspondiente a esa clase. Sin embargo, parece no favorecer las búsquedas que implican todas (o algunas) de las clases de la jerarquía indexada, ya que eso exigiría el recorrido indiscriminado de los árboles asociados a cada una de las clases.

6.1.1.2 CH-Tree

6.1.1.2.1 Adecuado para consultas que implican varias clases de la jerarquía

Si el predicado de una consulta implica únicamente una clase en la jerarquía de clases, el SC se comporta mejor que el *CH-Tree*. Por el contrario, si se implican todas las clases de la jerarquía (o al menos dos) en la consulta, el *CH-Tree* se comporta mejor que un conjunto de índices SC. Con relación al tamaño de los índices generados por los dos métodos no muestran un claro ganador o perdedor entre el *CH-Tree* y el conjunto de índices SC.

6.1.1.3 H-Tree

6.1.1.3.1 Adecuado para consultas de recuperación en un número pequeño de clases de la jerarquía

El *H-Tree* se comporta mejor que el *CH-Tree* para consultas de recuperación, especialmente cuando en la consulta se referencia un número pequeño de clases de la jerarquía, ya que, el *CH-Tree* emplea la misma estrategia para la recuperación de datos de una clase simple que de una jerarquía de clases.

Para las consultas de actualización no hay estudios suficientes para concluir cual de los dos se comporta mejor.

6.1.1.4 *hcC-Tree*

6.1.1.4.1 Es menos dependiente de la distribución de los datos que *CH-Tree* y *H-Tree*

Los *hcC-Trees* son más fáciles de implementar, y de hecho más simples que los *H-Trees*, ya que no hay anidamiento de árboles. Además, los experimentos realizados atendiendo a la distribución de los valores clave en la jerarquía de clases para controlar el número de páginas accedidas muestran que *H-Trees* y *CH-Trees* son más dependientes de la distribución actual de los datos. Los *H-Trees* no tienen buen rendimiento para CHR y CHP, y los *CH-Tree* no tienen buen rendimiento para consultas SCR.

6.1.1.4.2 Se comporta mejor que el *H-Tree* cuando el número de clases en la jerarquía aumenta

Cuando el número de clases en una jerarquía crece el rendimiento del *H-Tree* se degrada para consultas CHP, ya que hay más saltos desde los *H-Trees* de las superclases a los *H-Trees* de las subclases. A su vez, el *CH-Tree* se comporta ligeramente mejor que el *hcC-Tree*.

6.1.2 Técnicas de jerarquía de agregación

6.1.2.1 *Path* (PX)

6.1.2.1.1 Permite resolver predicados anidados sobre todas las clases del camino

Un PX almacena instancias de caminos, y como se puede observar, puede ser empleado para evaluar predicados anidados sobre todas las clases a lo largo del camino.

6.1.2.2 *Nested* (NX)

6.1.2.2.1 Bajo costo en operaciones de recuperación

Con este índice la evaluación de un predicado para el camino P sobre un atributo anidado A(n) de la clase C(1) requiere el recorrido de un único índice, por tanto, el costo de resolver un predicado anidado es equivalente al costo de resolver el predicado sobre un atributo simple C(1).

6.1.2.3 Multiindex (MX)

6.1.2.3.1 Bajo costo en operaciones de actualización

Para consultas de actualización, el que mejor se comporta es el MX, seguido del PX, y finalmente el NX. De hecho, la razón de dividir un camino en subcaminos, es principalmente, para reducir los costos de actualización del NX o del PX, pero permitiendo a la vez una recuperación eficiente.

6.1.2.3.2 Alto costo en operaciones de recuperación

Si se examina el rendimiento con relación a consultas de recuperación, el NX es el que mejor se comporta, ya que el PX tiene un gran tamaño, pero éste último se comporta bastante mejor que el MX, ya que éste necesita acceder a varios índices.

6.1.2.3.3 PX es el mejor teniendo en cuenta todas las operaciones

Para concluir, se puede decir que teniendo en cuenta las diferentes clases de operaciones (recuperación, actualización, inserción y borrado) el PX se comporta mejor que las otras dos estructuras.

6.1.2.3.4 MX requerimientos de espacio intermedios entre NX y PX

Se puede concluir que en cuanto a requerimientos de espacio, el NX siempre tiene los requerimientos más bajos. El PX requiere siempre más espacio que el MX, excepto cuando hay pocas referencias compartidas entre objetos, o bien no hay ninguna. Esto es debido a que el mismo IDO puede aparecer en muchas instancias de camino, ocasionando que ese IDO sea almacenado muchas veces.

6.1.2.4 *Path Dictionary Index (PDI)*

6.1.2.4.1 Posibilita la existencia de muchos índices sobre atributos

Cuando hay un atributo indexado el PD se comporta mejor en cuanto a sobrecarga de almacenamiento que el PDI y este mejor que el *Path* (PX), pero cuando hay más de un atributo indexado, la sobrecarga de almacenamiento del PX se incrementa mucho más que el PD y el PDI. Los requerimientos de almacenamiento para el PD son constantes, ya que es suficientemente general para soportar todos los tipos de consultas anidadas. Para el PDI las necesidades de almacenamiento extra para crear los índices *attribute* son bajas en comparación con el PX.

6.1.2.4.2 El PDI tiene mejor rendimiento en la recuperación que el PX

El PDI presenta mejor rendimiento en operaciones de recuperación que el PX y además, el PX no puede emplearse para soportar consultas en las que la clase predicado es un ancestro de la clase objetivo. En general, el PDI y el PD se comportan mucho mejor que el PX cuando se considera una mezcla general de consultas anidadas.

6.1.2.4.3 El PX se comporta mejor en consultas de rango

El PX presenta un rendimiento en consultas de rango mejor que las otras dos opciones, y esto es debido a que los registros del nodo hoja del PX están ordenados basándose en los valores clave del atributo indexado, de forma que después de acceder al registro correspondiente al valor clave más bajo, se pueden devolver los IDOs de los objetos que interesan recorriendo los nodos hojas hasta que se alcance el valor del rango más alto. Sin embargo, el PDI tiene que acceder a las s-expresiones en el PD para devolver los IDOs de los objetos después de obtener las direcciones de las s-expresiones. El PD tiene el peor rendimiento, ya que la evaluación de predicados está basada en acceder a los objetos en la clase predicado.

6.1.2.4.4 El PD se comporta mejor en actualización que el PX

En operaciones de actualización el PD se comporta mejor que el PDI, y cualquiera de ellos se comporta mejor que el PX.

6.1.2.5 *Join Index Hierarchy* (JIH)

6.1.2.5.1 JIH-C requiere más espacio y más costo de actualización

El algoritmo para la construcción de la jerarquía JIH-C es el mismo que el anterior pero sin necesidad de seleccionar los índices auxiliares, ya que aquí son materializados todos los índices. La recuperación podría ser más rápida si se emplea una jerarquía completa que si se emplea una parcial que requiere acceder a nodos virtuales que no han sido materializados directamente. Sin embargo, una jerarquía completa requiere más espacio de almacenamiento y más costo de actualización que una parcial (aunque los algoritmos son los mismos).

6.1.2.5.2 JIH-B se comporta peor que JIH-P y que JIH-C

Una JIH-B es construida y actualizada de forma más sencilla que las JIH-P, ya que JIH-B es una jerarquía degenerada, y por tanto, no necesita una propagación de las actualizaciones hacia arriba.

Sin embargo, la navegación entre dos clases $C(i)$ y $C(i+l)$ en una JIH-B requiere el recorrido de una secuencia de l índices join base: $Jl(i,i+1), \dots, Jl(i+l-1,i+l)$, a diferencia de un JIH-P que requiere el recorrido de uno o un número muy pequeño de Jl .

6.1.2.5.3 División del esquema de camino para esquemas de caminos largos

A medida que se incrementa la longitud de un esquema de camino se incrementa también el tamaño de la JIH y por supuesto su costo de actualización, y es por ello necesario acudir a una partición del esquema de camino en otros más pequeños y fácilmente manejables.

6.1.3 Técnicas de jerarquía de herencia y agregación

6.1.3.1 *Nested Inherited*

6.1.3.1.1 Apropriado cuando todos los atributos del camino son de valor-simple

Esta técnica es apropiada cuando todos los atributos en el camino indexado son simples; o cuando únicamente las primeras clases en el camino tienen atributos multi-valuados, mientras que todas las demás clases tienen atributos simples. En el resto de situaciones esta técnica tiene un costo superior a IMX.

6.1.3.2 *Inherited MultiIndex*

6.1.3.2.1 Apropiado cuando las consultas son principalmente sobre la última clase del camino indexado

El rendimiento del IMX es mejor que el del NIX cuando las consultas son principalmente sobre la última clase del camino indexado, por lo tanto, el NIX debe emplearse principalmente cuando el número de predicados anidados en la consulta es elevado.

6.2 Técnicas de comportamiento

6.2.1 Técnicas basadas en la invocación de métodos

6.2.1.1 Materialización de métodos

6.2.1.1.1 Apropiado para operaciones de búsqueda

El rendimiento de esta técnica en operaciones de búsqueda es muy eficiente, ya que el resultado de los métodos ya está precalculado o el valor de los métodos más utilizados es el que se encuentra precalculado, y no se necesita realizar la ejecución de el código que se encuentra encapsulado en el método ya que esto ocasionaría que se degradara el rendimiento de la búsqueda.

7 PRUEBAS DE RENDIMIENTO

7.1 Escenario de pruebas

Para la realización de las pruebas de rendimiento se utilizaron dos bases de datos; una base de datos relacional la cual es *Sql Server* y otra orientada a objetos que es *Oracle 9i*. A continuación se describen las características de cada una de las dos máquinas utilizadas. Las características se clasifican por *software* y *hardware*:

7.1.1 Servidor de base de datos relacional

7.1.1.1 *Hardware*

- Procesador *Pentium II* de 450 MHz.
- Memoria RAM 256 MB.
- Memoria Virtual: Tamaño inicial de 200 MB, tamaño máximo de 400 MB.
- Partición NTFS.
- Espacio de Disco Duro 4 GB.

7.1.1.2 Software

- **Sistema operativo**
- *Windows* 2000 Profesional.
- Paquete de servicio No. 3 para *Windows* 2000.

2. Base de datos

- *SQL Server* 2000 Edición Empresarial.

7.1.2 Servidor de base de datos orientada a objetos

7.1.2.1 Hardware

- Procesador *Pentium* II de 500 MHz.
- Memoria RAM 256 MB.
- Memoria Virtual: Tamaño inicial de 200 MB, tamaño máximo de 400 MB.
- Partición NTFS.
- Espacio de Disco Duro 4.75 GB.

7.1.2.2 Software

2. Sistema operativo

- *Windows* 2000 Profesional.
- Paquete de servicio No. 3 para *Windows* 2000.

3. Base de datos

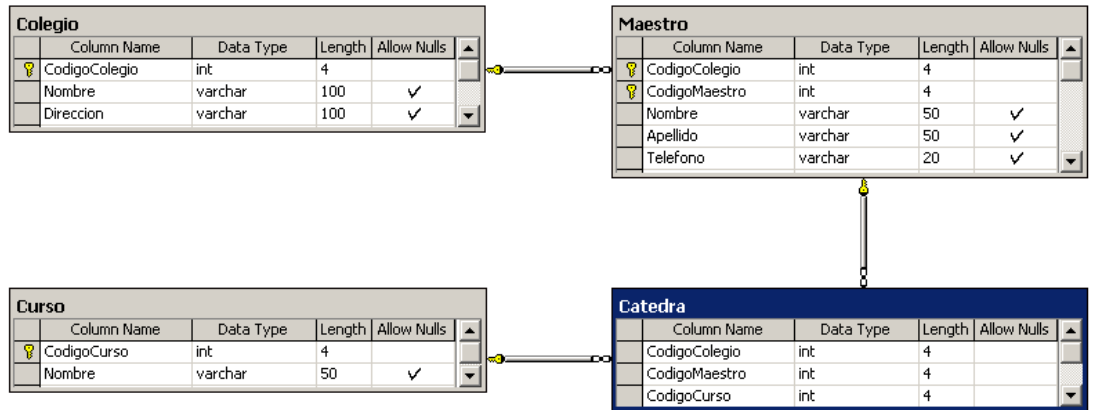
- *Oracle* 9i.

7.2 Componentes para la realización de pruebas

7.2.1 Base de datos relacional

Para la base de datos relacional se crearon dos tablas que se muestran en el modelo entidad relación, las dos tablas están unidas por una relación de uno a muchos la cual indica que para un colegio pueden existir muchos maestros.

Figura 18. Modelo entidad relación



El código *script* para la generación de las tablas presentadas anteriormente es el siguiente:

```
CREATE TABLE [dbo].[Catedra] (
    [CodigoColegio] [int] NOT NULL ,
    [CodigoMaestro] [int] NOT NULL ,
    [CodigoCurso] [int] NOT NULL ,
    [Fecha] [datetime] NOT NULL
) ON [PRIMARY]
GO
```

```
CREATE TABLE [dbo].[Colegio] (
    [CodigoColegio] [int] NOT NULL ,
    [Nombre] [varchar] (100) NULL ,
    [Direccion] [varchar] (100) NULL
) ON [PRIMARY]
GO
```

```
CREATE TABLE [dbo].[Curso] (
    [CodigoCurso] [int] NOT NULL ,
    [Nombre] [varchar] (50) COLLATE NULL
) ON [PRIMARY]
GO
```

```
CREATE TABLE [dbo].[Maestro] (
    [CodigoColegio] [int] NOT NULL ,
```

```

    [CodigoMaestro] [int] NOT NULL ,
    [Nombre] [varchar] (50) COLLATE NULL ,
    [Apellido] [varchar] (50) COLLATE NULL ,
    [Telefono] [varchar] (20) COLLATE NULL
) ON [PRIMARY]
GO

```

```

ALTER TABLE [dbo].[Colegio] WITH NOCHECK ADD
    CONSTRAINT [PK_Colegio] PRIMARY KEY CLUSTERED
    (
        [CodigoColegio]
    ) ON [PRIMARY]
GO

```

```

ALTER TABLE [dbo].[Curso] WITH NOCHECK ADD
    CONSTRAINT [PK_Curso] PRIMARY KEY CLUSTERED
    (
        [CodigoCurso]
    ) ON [PRIMARY]
GO

```

```

ALTER TABLE [dbo].[Maestro] WITH NOCHECK ADD
    CONSTRAINT [PK_Maestro] PRIMARY KEY CLUSTERED
    (
        [CodigoColegio],
        [CodigoMaestro]
    ) ON [PRIMARY]
GO

```

```

ALTER TABLE [dbo].[Catedra] ADD
    CONSTRAINT [FK_Catedra_Curso] FOREIGN KEY
    (
        [CodigoCurso]
    ) REFERENCES [dbo].[Curso] (
        [CodigoCurso]
    ),

```

```

CONSTRAINT [FK_Catedra_Maestro1] FOREIGN KEY
    (

```

```

        [CodigoColegio],
        [CodigoMaestro]
    ) REFERENCES [dbo].[Maestro] (
        [CodigoColegio],
        [CodigoMaestro]
    )
GO

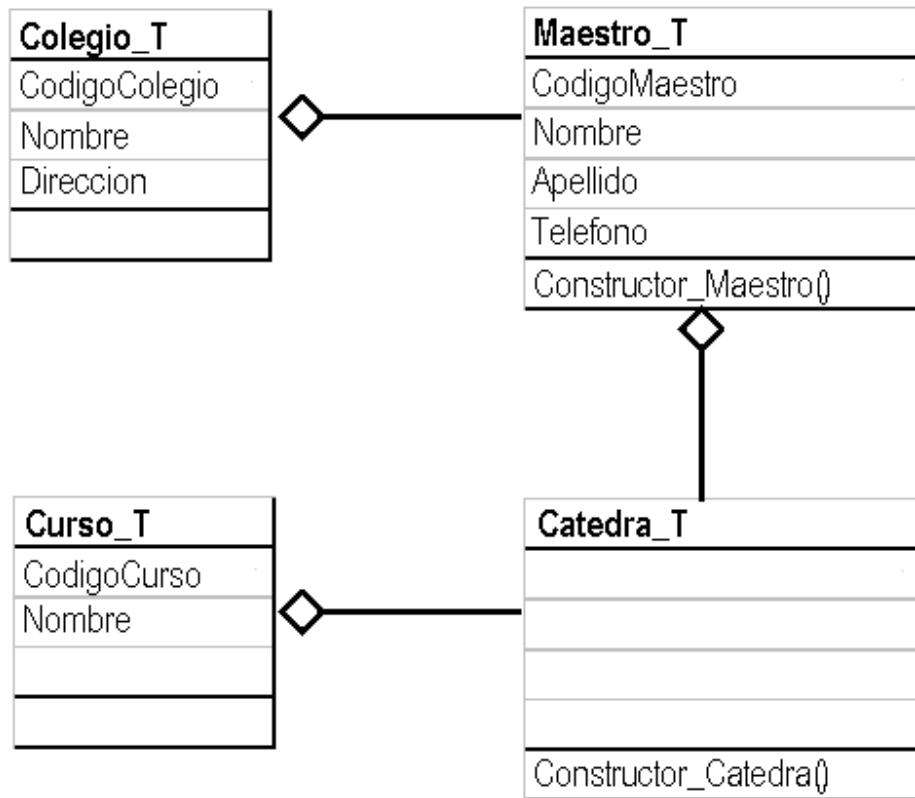
ALTER TABLE [dbo].[Maestro] ADD
    CONSTRAINT [FK_Maestro_Colegio] FOREIGN KEY
    (
        [CodigoColegio]
    ) REFERENCES [dbo].[Colegio] (
        [CodigoColegio]
    )
GO

```

7.2.2 Base de datos orientada a objetos

Para la base de datos orientada a objetos se crearon dos clases que se muestran en el modelo de clases, las dos clases están unidas por una relación de asociación de uno a muchos, la cual indica que para un colegio pueden existir muchos maestros.

Figura 19. Diagrama de clases



El código *script* para la generación de las clases presentadas en el diagrama anterior es el siguiente:

```

CREATE OR REPLACE TYPE "COLEGIO_T" AS OBJECT (
  "CODIGOCOLEGIO" NUMBER(10, 0),
  "NOMBRE" CHAR(100),
  "DIRECCION" CHAR(100))

CREATE TABLE COLEGIO OF COLEGIO_T;

CREATE OR REPLACE TYPE "MAESTRO_T" AS OBJECT (
  "REFCOLEGIO" REF "COLEGIO_T",
  "CODIGOMAESTRO" NUMBER(10, 0),
  "NOMBRE" CHAR(50),
  "APELLIDO" CHAR(50),
  "TELEFONO" CHAR(21),

```

```

MEMBER FUNCTION "CONSTRUCTOR_MAESTRO"
(PCODIGOCOLEGIO IN NUMBER,
 PCODIGOMAESTRO IN NUMBER,
 PNOMBRE IN VARCHAR2,
 PAPELLIDO IN VARCHAR2,
 PTELEFONO IN VARCHAR2)
RETURN "MAESTRO_T" )

CREATE OR REPLACE TYPE BODY MAESTRO_T IS
STATIC function constructor_Maestro
( pCodigoColegio number,
 pCodigoMaestro number,
 pNombre VARCHAR2,
 pApellido VARCHAR2,
 pTelefono VARCHAR2)
RETURN Maestro_t IS
vRefCol REF Colegio_t;
BEGIN
Select REF(c) INTO vRefCol
From Colegio c
Where c.CodigoColegio = pCodigoColegio;
return Maestro_t(vRefCol, pCodigoMaestro, pNombre, pApellido, pTelefono);
END;
END;

CREATE TABLE MAESTRO OF MAESTRO_T;

CREATE OR REPLACE TYPE "CURSO_T" AS OBJECT (
"CODIGOCURSO" NUMBER(10, 0),
"NOMBRE" CHAR(100))

CREATE TABLE CURSO OF CURSO_T;

CREATE OR REPLACE TYPE "CATEDRA_T" AS OBJECT (
"REFCURSO" REF "CURSO_T",
"REFMAESTRO" REF "MAESTRO_T",
MEMBER FUNCTION "CONSTRUCTOR_CATEDRA"
( PCODIGOCOLEGIO IN NUMBER,
 PCODIGOMAESTRO IN NUMBER,
 PCODIGOCURSO IN NUMBER)
RETURN "CATEDRA_T" )

CREATE OR REPLACE TYPE BODY CATEDRA_T IS
STATIC function constructor_Catedra

```

```

(PCODIGOCOLEGIO IN NUMBER,
PCODIGOMAESTRO IN NUMBER,
PCODIGOCURSO IN NUMBER)
RETURN Catedra_t IS
    vRefCur REF Curso_t;
    vRefMae REF Maestro_t;
BEGIN
    Select REF(c) INTO vRefCur
        From Curso c
        Where c.CodigoCurso = pCodigoCurso;
    Select REF(c) INTO vRefMae
        From Maestro c
        Where c.CodigoColegio = pCodigoColegio and c.CodigoMaestro = pCodigoMaestro;
    return Maestro_t(vRefCur, vRefMae);
END;
END;

CREATE TABLE CATEDRA OF CATEDRA_T;

```

7.2.3 Aplicación de pruebas

La aplicación para la realización de las pruebas se desarrollo con el lenguaje de programación Visual Basic versión 6.0.

Dentro de la aplicación se realiza la conexión a las dos bases de datos para realizar las inserciones, las consultas y las eliminaciones. Y al momento de realizar estas operaciones, dentro de la aplicación se toma el tiempo de inicio y el tiempo final en el que dura la operación y esté es utilizado para compararlo con los otros tiempos.

7.2.3.1 Ingreso y eliminación de información

Para el ingreso de la información a las entidades “Colegio” y “Maestro” se realiza por medio de dos ciclos anidados, el primer ciclo inserta el colegio y el otro ciclo anidado inserta los maestros para dicho colegio.

Luego para el ingreso de la información en la entidad de “Curso” solo se ingresaron 2 cursos básicamente, y para la entidad de “Cátedra” se ingresaron a los maestros con código impar con el curso 1 que es Matemática y a los maestros con código par el curso 2 que es Física.

Este mismo esquema es utilizado para el ingreso de la información tanto en la base de datos relacional como la base de datos orientada a objetos.

Las sentencias utilizadas para el ingreso de la información en la base de datos relacional es la siguiente:

```
Insert Into Colegio Values (1, 'Nombre Colegio 1', 'Dirección')
Insert Into Maestro Values (1, 1, 'Nombre Maestro 1 - 1', 'Dirección', 'Teléfono')
Insert Into Curso Values (1, 'Matematica' )
Insert Into Catedra Values (1, 1, 1)
```

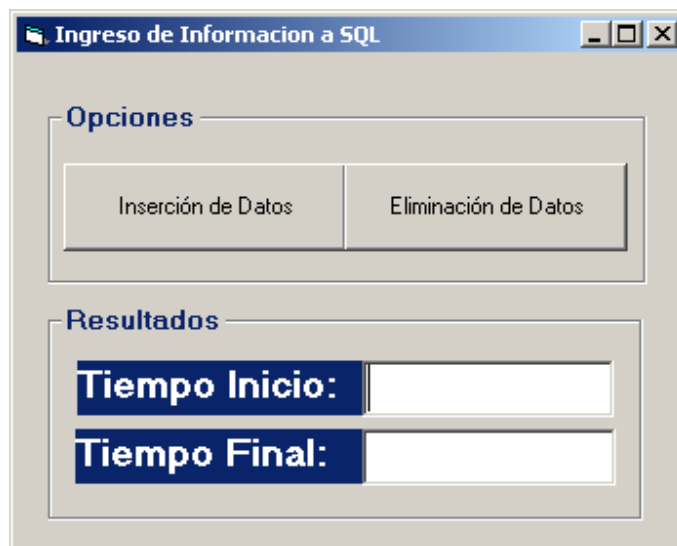
Las sentencias utilizadas para el ingreso de la información en la base de datos orientada a objetos es la siguiente:

```
Insert Into Colegio Values (1, 'Nombre Colegio 1', 'Dirección')
```

```
Insert Into Maestro Values (maestro_t.constructor_maestro(1,1,'Nombre Maestro 1 – 1','Dirección', 'Teléfono'))
Insert Into Curso Values (1, 'Matematica');
Insert Into Catedra Values (catedra_t.constructor_catedra(1,1,1));
```

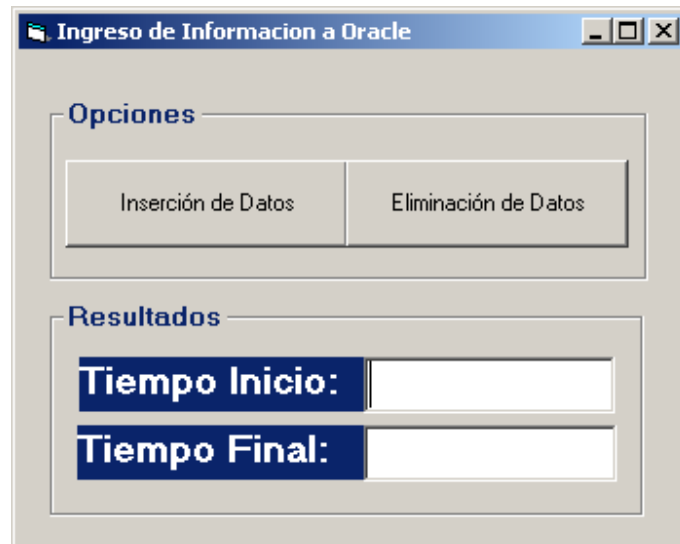
La forma para el ingreso y la eliminación a la base de datos relacional es la figura siguiente:

Figura 20. Forma de ingreso de información a SQL



La forma para el ingreso y la eliminación a la base de datos relacional es la figura siguiente:

Figura 21. Forma de ingreso de información a Oracle



7.2.3.2 Consulta de información

La consulta de la información se realiza por medio de la ejecución de un *query*, el cual realiza la búsqueda de la información necesaria y la muestra en pantalla. En este caso las consultas para una base de datos orientada a objetos son muy diferentes a las que se realizan dentro de la base de datos relacional.

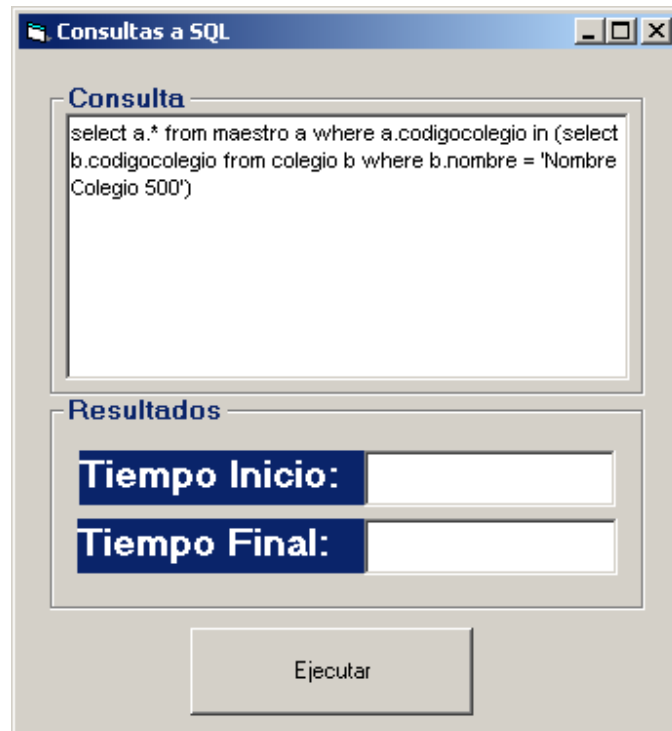
7.2.3.2.1 Consulta a una base de datos relacional

A continuación se muestran las consultas realizadas en las pruebas de rendimiento a la base de datos relacional.

```
1      select a.*
      from maestro a
      where a.codigocolegio in
            (select b.codigocolegio
             from colegio b
             where b.nombre = 'Nombre Colegio 500')
```

```
2      select a.Apellido, a.Nombre
      from maestro a, catedra b, curso c, colegio d
      where a.codigomaestro = b.codigomaestro and
            a.codigocolegio = b.codigocolegio and
            c.codigocurso = b.codigocurso and
            d.codigocolegio = a.codigocolegio and
            c.Nombre = 'Matematica' and
            d.Nombre = 'Nombre Colegio 500'
```

Figura 22. Forma para consultas a SQL



7.2.3.2.2 Consulta a una base de datos orientada a objetos

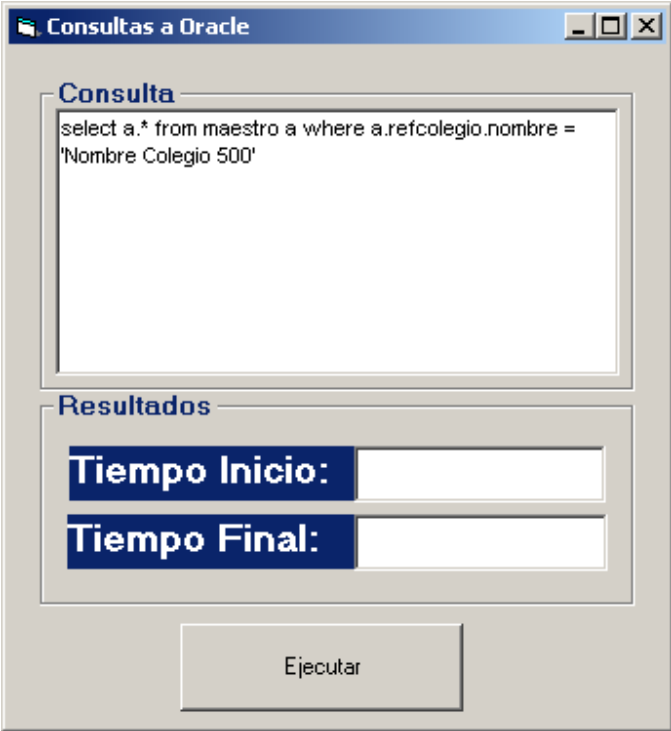
A continuación se muestran las consultas realizadas en las pruebas de rendimiento a la base de datos orientada a objetos.

- 1 `select a.*
from maestro a
where a.refcolegio.nombre = 'Nombre Colegio 500'`

- 2 `select a.refmaestro.apellido, a.refmaestro.nombre
from catedra a`

where a.refcurso.nombre = 'Matematica' and
a.refmaestro.refcolegio.nombre = 'Nombre Colegio 500'

Figura 23. Forma para consultas a Oracle



The screenshot shows a window titled "Consultas a Oracle" with a standard Windows-style title bar. Inside the window, there are two main sections. The top section, labeled "Consulta", contains a text area with the following SQL query: `select a.* from maestro a where a.refcolegio.nombre = 'Nombre Colegio 500'`. The bottom section, labeled "Resultados", contains two input fields for timing: "Tiempo Inicio:" and "Tiempo Final:". Below these fields is a button labeled "Ejecutar".

7.3 Resultados de las pruebas

Para la realización de las pruebas se tomaron tres muestras con diferentes cantidades de datos. Las cantidades de registros se detallan en la siguiente tabla:

Tabla I. Detalle de muestras

Muestra	Colegios		Maestros	
1	1,000	registros	20,000	registros
2	10,000	registros	200,000	registros
3	100,000	registros	1,000,000	registros

7.3.1 Pruebas de ingreso

A continuación se muestra la tabla con el resultado de las pruebas realizadas.

Tabla II. Resultado de pruebas de ingreso con índice (Formato HH:MM:SS)

Muestra	BD Relacional	BD Orientada a Objetos
1	00:00:44	00:03:55
2	00:06:39	00:17:54
3	01:05:45	01:59:48

Figura 24. Pruebas de ingreso con índice (Formato HH:MM:SS)

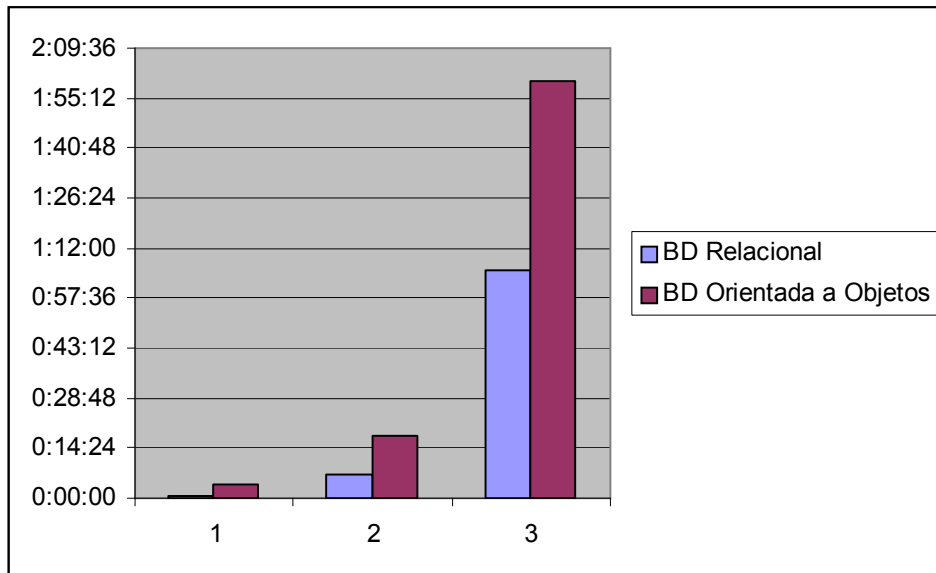
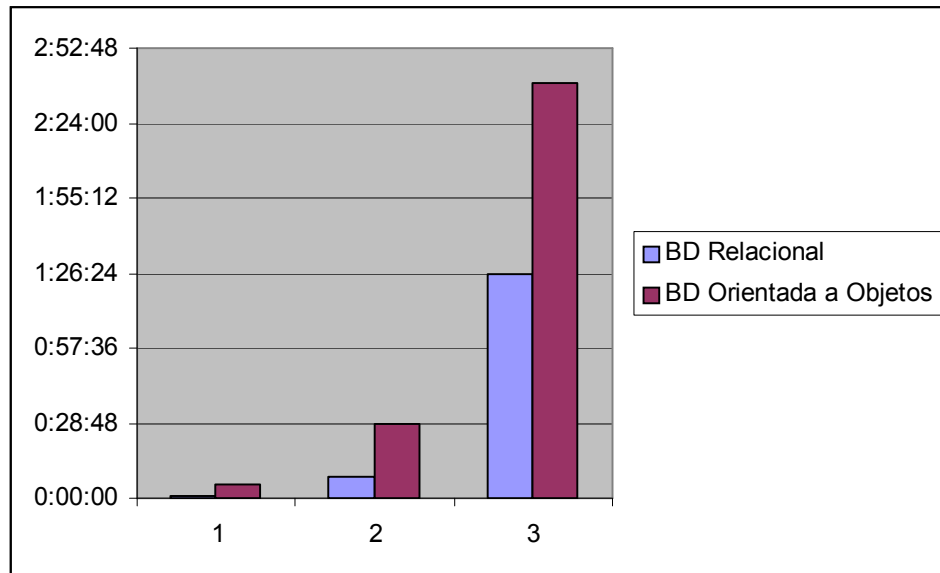


Tabla III. Resultado de pruebas de ingreso sin índice (Formato HH:MM:SS)

Muestra	BD Relacional	BD Orientada a Objetos
1	00:01:05	00:05:05
2	00:08:27	00:28:44
3	01:25:58	02:39:07

Figura 25. Pruebas de ingreso sin índice (Formato HH:MM:SS)



7.3.2 Pruebas de consultas

A continuación se muestra la tabla con el resultado de las pruebas realizadas.

**Tabla IV. Resultado de pruebas de consulta con índice
(Formato HH:MM:SS)**

Muestra	BD Relacional	BD Orientada a Objetos
1	00:00:03	00:00:01
2	00:00:14	00:00:03
3	00:00:49	00:00:04

Figura 26. Pruebas de consulta con índice (Formato HH:MM:SS)

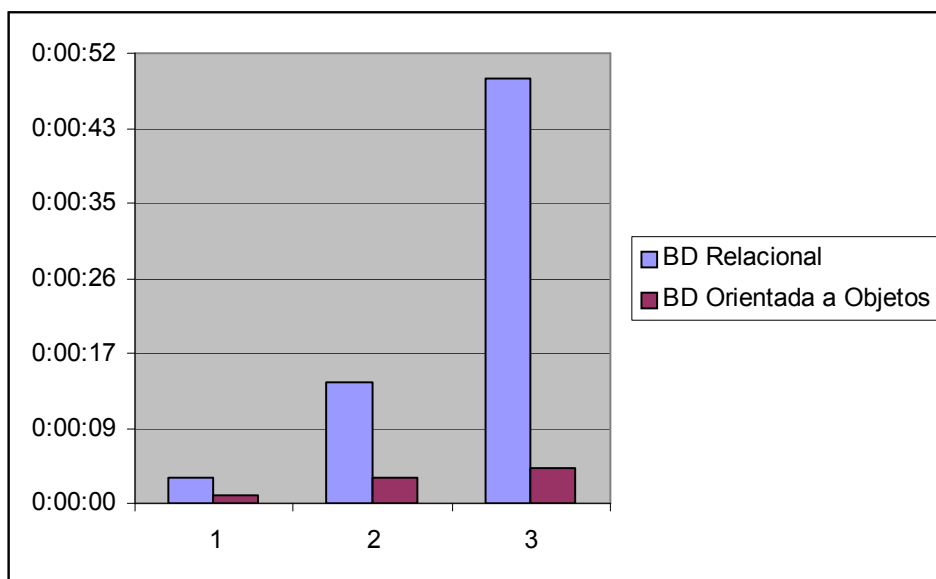
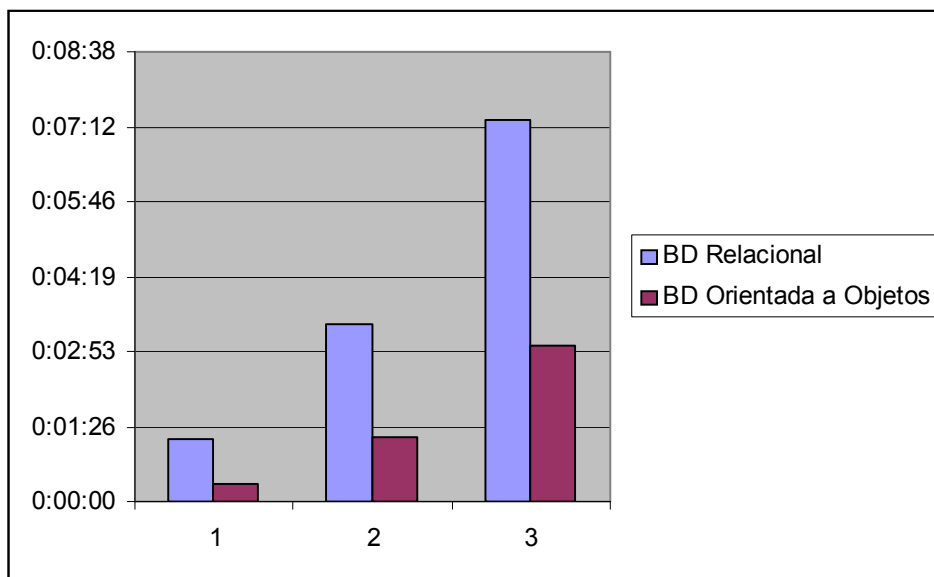


Tabla V. Resultado de pruebas de consulta sin índice (Formato HH:MM:SS)

Muestra	BD Relacional	BD Orientada a Objetos
1	00:01:12	00:00:21
2	00:03:24	00:01:13
3	00:07:19	00:02:59

Figura 27. Pruebas de consulta sin índice (Formato HH:MM:SS)



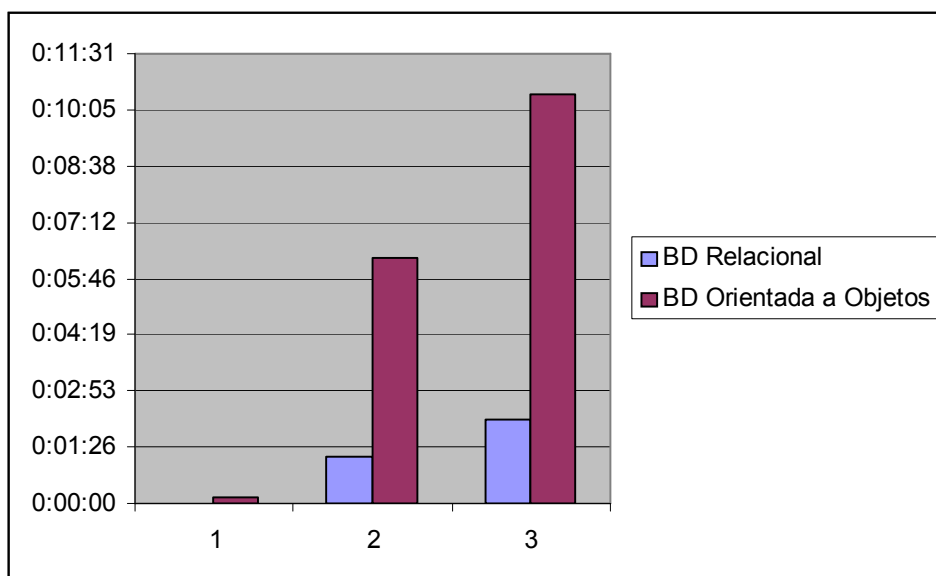
7.3.3 Pruebas de eliminación

A continuación se muestra la tabla con el resultado de las pruebas realizadas.

**Tabla VI. Resultado de pruebas de eliminación con índice
(Formato HH:MM:SS)**

Muestra	BD Relacional	BD Orientada a Objetos
1	00:00:01	00:00:09
2	00:01:12	00:06:18
3	00:02:08	00:10:29

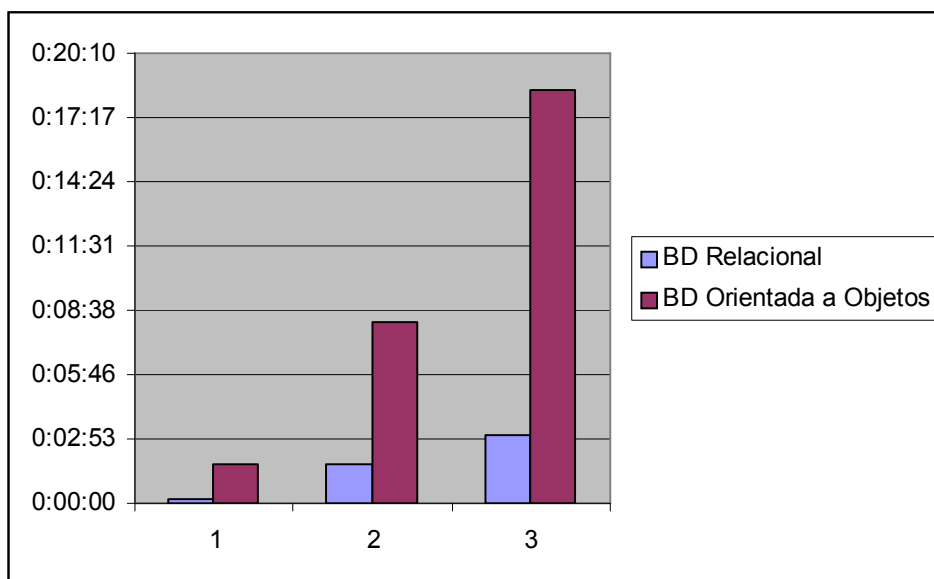
Figura 28. Pruebas de eliminación con índice (Formato HH:MM:SS)



**Tabla VII. Resultado de pruebas de eliminación sin índice
(Formato HH:MM:SS)**

Muestra	BD Relacional	BD Orientada a Objetos
1	00:00:11	00:01:46
2	00:01:47	00:08:09
3	00:03:01	00:18:31

Figura 29. Pruebas de eliminación sin índice (Formato HH:MM:SS)



7.4 Análisis de los resultados

Gracias a los resultados de estas pruebas, se puede comprobar que el rendimiento en la búsqueda de los datos es más rápido en una base de datos orientada a objetos, ya que internamente los datos están asociados por medio de una estructura de árbol que tiene como objetivo principal referenciar a los objetos que contienen una relación de asociación.

En la consulta a la base de datos orientada a objetos no se debe de hacer la relación entre las dos clases dado a que está ya existe, y por lo tanto, se puede acceder a ella por medio de la clase.

En base a los resultados se puede apreciar que existe un mal rendimiento en el ingreso y eliminación de la información en la base de datos orientada a objetos a diferencia a la base de datos relacional, esto se debe a que en el ingreso además de insertar los datos también se ingresa la referencia entre las dos clases y esto es lo que lo hace más lento; al igual que en la eliminación se realizan las mismas acciones que en el ingreso, por lo que la eliminación tiene un mal rendimiento en comparación con la base de datos relacional.

CONCLUSIONES

- Los lenguajes de programación actuales se acoplan completamente con los sistemas de administración de bases de datos orientados a objetos, lo cual da un desarrollo de aplicaciones rápidas y flexibles.
- Los índices utilizados en las bases de datos orientadas a objetos hacen que la búsqueda de la información sea más rápida en grandes volúmenes de información a diferencia de las bases de datos relacionales.
- Debido a que las búsquedas de información son más rápidas, se disminuye la velocidad en el ingreso y eliminación, por lo que en este aspecto las bases de datos relaciones son las eficientes.
- El estudio detallado de las técnicas de indexación da gran ayuda al momento de agregar índices a un objeto dentro de la base de datos, ya que dependiendo del tipo de jerarquía que se tenga se puede seleccionar el tipo de índice que uno considera más adecuado.

- Las bases de datos orientadas a objetos son una buena herramienta para una empresa que posee gran cantidad de información y que requiera tener acceso rápido sin importar el bajo rendimiento en el ingreso y la eliminación.

RECOMENDACIONES

- Al momento de elegir la base de datos a utilizar se debe de tomar en cuenta no solo la cantidad de datos; sino que además, tomar en cuenta el tiempo que se dispone al momento de necesitar la información y el tiempo para insertar y eliminar la información.

- Además se debe considerar que el paradigma de programación más reciente es el orientado a objetos y esto demanda la utilización de un administrador de bases de datos orientado a objetos.

- Se debe de elegir el tipo de indexación adecuado, dependiendo del tipo de jerarquía que se tenga relacionando dos o más objetos.

- Debido al creciente uso de la información dentro de las empresas se recomienda la utilización de las bases de datos orientadas a objetos, dado a que éstas son más eficientes conforme el volumen de la información crezca.

- Se debe de considerar la cantidad de índices que se le colocan a los objetos, ya que dependiendo de la cantidad, ese es el espacio utilizado para el mantenimiento del mismo.

BIBLIOGRAFÍA

- Bertino Elisa y Lorenzo Martino. **Sistemas de bases de datos orientadas a objetos. conceptos y arquitecturas**. Addison-Wesley, 1993. Traducción Diaz de Santos, 1995.
- Bertino, Elisa y otros. **Técnicas de indexación para sistemas avanzados de bases de datos**. Norteamérica: Editorial Kluwer, 1997.
- GemStone Systems, Inc, www.gemstone.com, Internet noviembre 2002.
- Oracle Corporation, www.oracle.com, Internet octubre 2002.
- Rojas Morales, Claudia Liceth. Arquitectura de las bases de datos orientadas a objetos, Tesis Ingeniería en Ciencias y Sistemas. Guatemala, Universidad de San Carlos de Guatemala, Facultad de Ingeniería, 1997.