UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ACATLAN"

ALMACENAMIENTO FISICO, LOGICO Y ACCESO A DATOS A TRAVES DE UN SISTEMA MANEJADOR DE BASES DE DATOS.

T E S I N A

QUE PARA OBTENER EL TITULO DE

LIC. MATEMATICAS APLICADAS Y C O M P U T A C I O N

P R E S E N T A

JUDITH NIETO ZAMACONA

UNAM
CAMPUS ACATLÁN

ASESOR: M. EN C. JUDITH JARAMILLO LOPEZ

MARZO 2003

AGRADEZCO

A mi ASESORA por su apoyo incondicional, su tiempo y dedicación a este trabajo.

A mi novio, Oscar por todo el amor que me da para sobresalir... este triunfo también es tuyo ya que fuiste mi fuente de inspiración para titularme. Te doy las gracias con el corazón.

A mis padres por su apoyo y paciencia.

A mis amigos por los ánimos infundados en el transcurso de la carrera.

I N D I C E

INTRODUCCIÓN

El mundo de las bases de datos es tan amplio, que se divide para facilitar su estudio. Por tanto, aquellas personas interesadas en este tema deben tener conocimientos específicos acerca de todos y cada uno de los aspectos que intervienen en el manejo de las mismas. Previo a la profundización de dichos aspectos se debe contar con conocimientos básicos acerca de los fundamentos y del entorno necesarios para su óptimo funcionamiento.

La elección del tema que aborda este trabajo se origina del poco o casi nulo conocimiento que poseen aquellas personas que se dedican al desarrollo de aplicaciones que explotan una base de datos, y que desconocen lo que hay detrás de dichas aplicaciones.

Uno de los conocimientos básicos es el **almacenamiento físico, lógico y acceso a datos** en una base de datos, proceso que lleva a cabo un **sistema manejador de bases de datos**. El propósito del presente trabajo es dar a conocer todos y cada uno de los factores que intervienen en este proceso destacando el papel tan significativo que desempeña un sistema manejador de base de datos en el flujo de una solicitud de información a la base de datos.

El tema central de este trabajo se basa en la búsqueda por conocer y comprender lo que hay detrás de una sentencia SQL en la que expresamos una petición en la cual va inmersa una solicitud de información específica a través de *un sistema manejador de base de datos relacional* y como resultado nos devuelve un desplegado de la información que satisface dicha petición, tema que se aborda con conocimientos teóricos y prácticos.

Consta de cuatro capítulos, el primero, contiene conocimientos muy específicos de los sistemas manejadores de bases de datos, ya que éstos junto con otros elementos son los que en realidad manejan el aspecto que se está considerando; los capítulos centrales hacen referencia a la teoría que existe para llevar a cabo el almacenamiento, acceso y organización de los datos, y por último el capítulo cuatro hace una comparación entre la teoría que existe y lo que un sistema manejador de bases de datos relacional (*ORACLE*) comercial utiliza para llevar a cabo el almacenamiento físico y lógico de los datos.

CAPITULO I

CONCEPTOS DE UN SISTEMA MANEJADOR DE BASES DE DATOS

Uno de los propósitos de este capítulo primeramente es dar a conocer aspectos específicos de los sistemas manejadores de bases de datos y en segundo lugar mostrar los mismos aspectos pero desde un punto de vista relacional, es decir, mostrar las bondades que tienen los sistemas manejadores de bases de datos relacionales. Por tanto la teoría presentada de éstos es explicada paralelamente con aspectos propios de los sistemas manejadores de bases de datos relacionales.

Se abordan aspectos tales como la definición, una breve historia que incluye los diferentes enfoques que han surgido, arquitectura, áreas en que se dividen para facilitar el estudio de los mismos y la definición de los elementos con los que interactúan para lograr eficazmente el manejo de la información. Por último se aborda la organización y aspectos fundamentales de los sistemas manejadores de bases de datos relacionales. Estos y otros aspectos son necesarios para comprender el contenido de la información de capítulos posteriores.

DEFINICIÓN: Un ***sistema manejador de bases de datos*** (DBMS, *Database Management System*) es el software que permite manejar los datos contenidos en la base de datos. Controla el acceso, mantiene la seguridad e integridad de la información así como permite definir la estructura de la misma. En otras palabras es una colección de procedimientos, funciones, procesos, y programas que son usados para facilitar el diseño, implementación y gestión de una base de datos.

Historia de los sistemas manejadores de bases de datos.

Hasta antes de la década de los 60's la información se tenía organizada en archivos, así cada programador definía su propia organización, los métodos de acceso, su estructura y todo lo concerniente a los datos. Esto causaba repetición e inconsistencia de la información. A partir de estos años se perciben los DBMS desde un enfoque distinto.

Los inconvenientes antes mencionados y otros más fueron la causa que motivó a Charles Bachman a diseñar el primer DBMS a principios de la década de los 60's que

fue llamado *"Integrated Data Store"*, siendo éste la base para crear el modelo de datos en red.

A finales de la misma década, IBM desarrolla IMS (*Information Management System*), siendo éste el primer DBMS comercial y que actualmente se sigue utilizando. Con IMS se dio inicio a la construcción del modelo de datos jerárquico. IMS estaba basado en un sistema de archivos y permitía el almacenamiento de grandes cantidades de datos. Sin embargo este DBMS no contaba con un sistema de recuperación de datos si estos no eran previamente respaldados, así como tampoco se controlaba el que dos usuarios pudieran modificar el mismo archivo al mismo tiempo.

En 1970 el Dr. Edgar F. Codd, miembro del Laboratorio de investigación IBM, propone una nueva representación de los datos llamada *"modelo relacional"*. Este modelo proponía que las bases de datos deberían verse como datos organizados en **relaciones** llamadas tablas, sobre las cuales se aplican los operadores del álgebra y del cálculo relacional.

Los sistemas manejadores de bases de datos relacionales (RDBMS, *Relational Database Management System*) contribuyeron fuertemente al desarrollo de las bases de datos pues proponían ser una herramienta efectiva en el manejo de las bases de datos ya que implementaban el álgebra y el cálculo relacional de una manera sencilla mediante un lenguaje de consulta estructurado (SQL, *Structured Consulta Language*). Basados en este modelo surgen: las primeras versiones de **ORACLE** hasta antes de ORACLE 8.x, este producto es de la compañía que lleva el mismo nombre, **OPENINGRES** de Computer Associates, **SYSTEM 10/11** de Sybase, **DYNAMICSERVER** de Informix, **DB/2** de IBM, etc.

En los 90's la tecnología siguió avanzando surgiendo así los sistemas manejadores de bases de datos objeto-relacionales (ORDBMS, *Object-Relational Data Base Management System*) que están basados en el modelo relacional pero se apoyan en los pilares de la Programación Orientada a Objetos. En esta categoría podemos mencionar a **ORACLE 8.x** de Oracle, **UNIVERSAL SERVER ILUSTRA** de Informix, **UNIVERSAL DATABASE DB/2 EXTENDERS** de IBM, **UNISQL/X** de UniSQL, **OSMOS** de Unisys, etc.

Existe también otra categoría, los sistemas manejadores de bases de datos orientados a objetos (OODBMS, *Object-Oriented Database Management System*) los cuales exclusivamente se basan en el paradigma Orientado a Objetos, entre estos se encuentran **JASMINE** de Computer Associates, **GEMSTONE** de Gemstone, **O2** de ORION, **OBJECT STORE** de Object Design, **OBJECTIVITY/DB** de Objectivity, **VERSANT OODBMS** de Versant, etc.

Actualmente los DBMS van en camino hacia la manipulación de imágenes y de video.

Las siguientes líneas abordan la teoría de los DBMS explicada paralelamente con aspectos propios de los RDBMS.

Arquitectura de un DBMS

La arquitectura a la que se hace referencia es un modelo estándar llamado *"arquitectura **ANSI/SPARC**"* y es llamado así debido a que fue propuesto por la ANSI/SPARC Study Group on Data Base Management System (*Grupo de estudio sobre sistemas manejadores de bases de datos*). Dicha arquitectura puede ser utilizada para cualquier DBMS ya sea relacional o algún otro. En capítulos posteriores se aborda la arquitectura de un RDBMS específicamente.

La arquitectura ANSI/SPARC de un sistema manejador de bases de datos está dividida en tres niveles [DAJC1986]:

1. Nivel interno
2. Nivel conceptual
3. Nivel externo

Nivel interno

El nivel interno, es la representación más cercana al almacenamiento físico. Se enfoca a las estructuras de datos que sirven para almacenar y organizar la información. Este nivel tiene asociado un *esquema interno*, el cual permite describir los datos exactamente como se encuentran almacenados en la computadora.

Básicamente en el nivel interno se definen los diferentes tipos de archivos en los que se almacenan los datos, se especifica si cada archivo tiene asociado un índice, la manera de acceder a los archivos que contienen los datos, etc.

Este primer nivel, es el que posteriormente se tratará con detenimiento primeramente dando a conocer la teoría que está involucrada (capítulo II) en la construcción de este nivel y posteriormente haciendo referencia al almacenamiento, organización y acceso de los datos en una base de datos montada en un RDBMS, siendo específicamente sobre ORACLE (capítulos III y IV).

Nivel conceptual

El nivel conceptual representa el nivel más alto de información de la base de datos, y se describe a través del *esquema conceptual*. Aquí es donde se definen todos los datos que serán almacenados, esta información se obtiene a partir de la definición de las famosas "reglas del negocio".

Nivel externo

En el nivel externo se maneja la información desde el punto de vista de cada usuario, se describe a través de un *esquema externo*. En otras palabras, es el conjunto de percepciones individuales de los datos de la base de datos. Cada vista individual o percepción se llama *subesquema o vista* [PARU1997].

La siguiente figura muestra un diagrama con los tres niveles y la relación existente entre éstos.

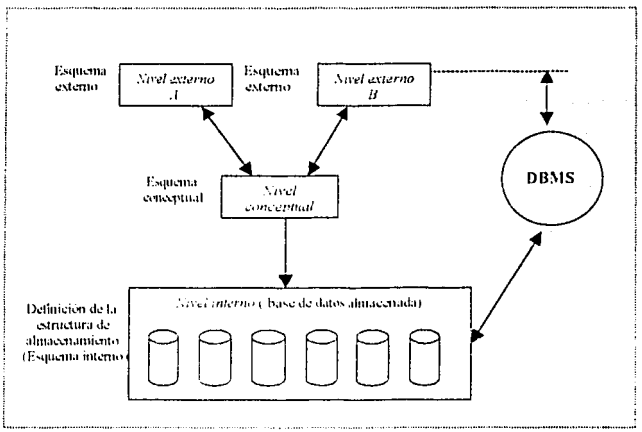


Figura 1.1: Diagrama que representa la conectividad del DBMS con los tres niveles de la arquitectura ANSI/SPARC.

El siguiente cuadro muestra los tres niveles ejemplificados basados en una base de datos de una liga de básquetbol. La ejemplificación del esquema interno está basada en el modelo relacional específicamente en ORACLE.

ESQUEMA EXTERNO

Subesquema 1: Jugadores con un peso mayor a 100 Kg.: jugador-peso, jugador-equipo.
Subesquema 2: Partidos ganados por equipo: equipo-local, equipo visitante, resultado.
Subesquema 3: Equipos con jugadores zurdos: equipo-nombre.

ESQUEMA CONCEPTUAL

EQUIPOS: Equipo-nombre, equipo-capitan, equipo-entrenador, etc.
JUGADORES: jugador-edad, jugador-altura, jugador-peso, etc.
PARTIDOS: Equipo-local, equipo-visitante, fecha, cancha, arbitros, etc.

Reglas de negocio:
1.- Un jugador solo pertenece a un equipo.
2.- Como maximo se permiten 2 jugadores de mas de 100 kg. por equipo.
3.- Como maximo se permiten 2 entrenadores por equipo, etc.

ESQUEMA INTERNO

TABLA	*TABLESPACE*	*DATAFILES*	*ATRIBUTOS*	*TIPO-DATOS*
Equipo	Basquetbol	Deporte, Disciplina	Eq-nombre, eq-capitan,	Varchar2(15), Varchar2(20),
Jugador	Basquetbol	Deporte, Disciplina	Ju-edad, Ju-altura	Number(2), Number(3,2)
Partido	Basquetbol	Deporte, Disciplina	Eq-local, Eq-visitante	Varchar2(15), Varchar2(15),

Cuadro 1.1: Muestra cada uno de los niveles de la arquitectura ANSI desde el punto de vista del manejador ORACLE.

A partir de la arquitectura ANSI/SPARC se puede hacer una división de las áreas de estudio de los sistemas manejadores de bases de datos para facilitar la comprensión y entendimiento de todos y cada uno de los aspectos que conforman a los mismos.

Áreas de estudio de un DBMS

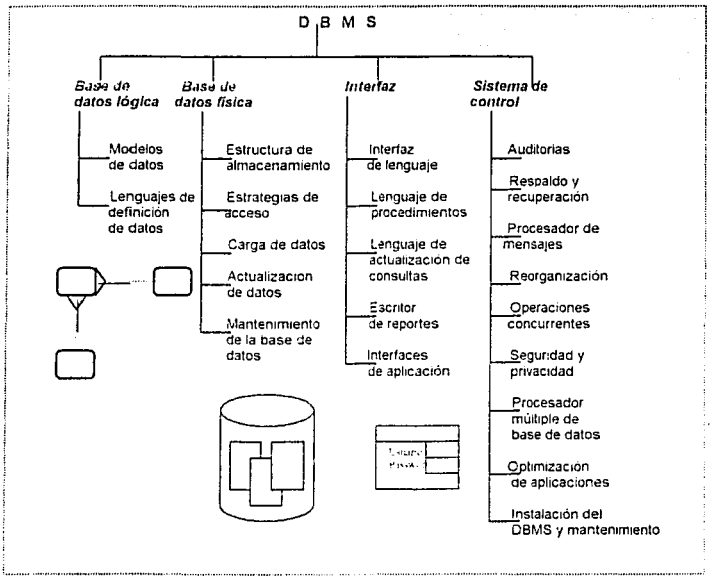
Debido a la complejidad y a la gran extensión que representan los sistemas manejadores de bases de datos como tema de estudio, se han dividido en cuatro grandes áreas:

I. Base de datos lógica,
II. Base de datos física,
III. Interfaz, y
IV. Sistema de control

Cada una de estas áreas tiene un equivalente en la arquitectura estándar ANSI/SPARC, así por ejemplo el área de base de datos lógica es equivalente al nivel conceptual; el

área de base de datos física atiende al nivel interno y las últimas dos áreas que son el área de interfaz y el área de sistema de control abordan aspectos propios del nivel externo.

EL siguiente diagrama muestra los aspectos más sobresalientes que conforman cada una de las áreas.



Cuadro 1.2: Muestra los aspectos importantes de cada una de las áreas en que se divide un DBMS para su estudio.

Base de datos lógica

Esta área abarca el análisis que se debe realizar previo a la creación de una base de datos en su parte lógica, lo que implica principalmente la especificación de las "reglas del negocio" las cuales son las condiciones que deben respetarse para la creación de la base de datos. El desarrollo de esta área se ve reflejado en un modelo de datos, el cual plasma todas los aspectos antes mencionados.

Refiriéndonos a un RDBMS estaríamos en la etapa de definir el "*modelo conceptual*", el cuál es un modelo de datos que contiene las "reglas del negocio" y se representa mediante la definición del modelo Entidad-Relación el cuál es un diagrama que estructura a los datos como tablas (entidades) y que define relaciones de acuerdo al contenido de las mismas. Un segundo aspecto que se considera parte de esta área es el lenguaje de definición de datos a través del cuál se lleva a cabo la implementación de la información que aporta el modelo de datos, nos referimos al lenguaje estructura de consulta SQL.

Base de datos física

Esta área comprende el manejo del almacenamiento de los datos, distribución de la información dentro de los dispositivos físicos de almacenamiento y acceso a los datos.

Al estudiar esta área se hace enfoque a las estructuras que utiliza cada DBMS para organizar los archivos que contienen los datos almacenados en la base de datos, incluyendo las técnicas de acceso a la información y las técnicas auxiliares de indexación para el acceso rápido a los datos.

Interfaz

Hace referencia a todo lo involucrado con la consulta de los datos así como la manera de seleccionar y retirar la información que es requerida. Se cuentan como parte de esta área los programas que sirven como interfaz para extraer información, los programas que hacen reportes de la misma, etc. Como parte de las herramientas que se utilizan en el manejo de la información, se encuentran las interfaces creadas en lenguajes de programación como C, C++, Delphi, etc., los cuales son utilizados para desarrollar aplicaciones complejas que se conectan con la base de datos.

Sistema de control

Esta área provee protección a la base de datos y a sus aplicaciones. De una manera muy general esta división se encarga de:

- Vigilar las transacciones
- Proveer ayuda en línea para detectar errores del DBMS
- Permite recuperar información a partir de respaldo
- Soportar la concurrencia de múltiples transacciones
- Mantener la seguridad de los datos

En cada uno de los niveles de la arquitectura ANSI/SPARC intervienen ciertos elementos que desempeñan un papel muy importante en el funcionamiento de todos los niveles en conjunto. El siguiente tema menciona los elementos que deben estar presentes para el buen funcionamiento de los DBMS.

Elementos que interactúan con el DBMS

Para realizar todas las funciones que es capaz de realizar un sistema manejador de base de datos requiere de un entorno propicio para ello en el cual hay elementos fundamentales como el Sistema Operativo, las aplicaciones mediante las cuales se explota la información contenida en la base de datos, etc.

La siguiente figura muestra la ubicación de cada uno de los elementos que interactúan con el DBMS.

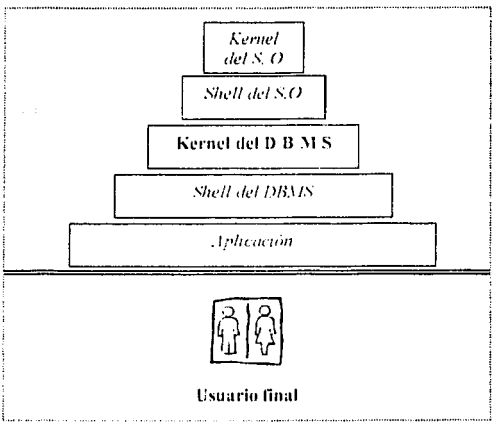


Figura 1.2 Muestra el entorno de un DBMS

En primer lugar se ubica el **kernel del Sistema Operativo** (S.O), este elemento es el más importante en el entorno de un DBMS ya que es éste quién en realidad maneja el almacenamiento físico de los datos contenidos en la base de datos por supuesto recibiendo instrucciones del DBMS. El sistema operativo es el encargado de manejar los dispositivos físicos de una computadora tales como memoria principal, memoria volátil, disco duro, etc.

El DBMS se instala sobre el sistema operativo es por ello que cuando se escoge un DBMS específico la selección se hace tomando en cuenta las características del sistema operativo.

Para administrar a los usuarios, el sistema operativo se vale de un **Shell de sistema operativo,** el cuál consiste en una serie de comandos que sirven para manejador el sistema operativo. La figura 1.3 muestra la relación que existe entre el sistema operativo y el DBMS.

En tercer lugar se ubica el **kernel del DBMS,** el cuál auxiliado por los demás elementos es capaz de cumplir el objetivo para el cual fue diseñado.

En cuarto lugar se encuentra el **Shell del DBMS,** es decir el intermediario entre el usuario y el DBMS, SQL ocupa este lugar en el modelo relacional ya que mediante los comandos propios de éste se lleva a cabo la explotación de información contenida en la base de datos.

En último lugar se encuentran las **aplicaciones o interfaces,** las cuales facilitan al usuario final extraer información de la base de datos, este usuario es totalmente ajeno al manejo de la información por lo que requiere una aplicación programada en algún lenguaje visual o no visual para acceder a la información sin recurrir a los comandos de algún lenguaje de consulta.

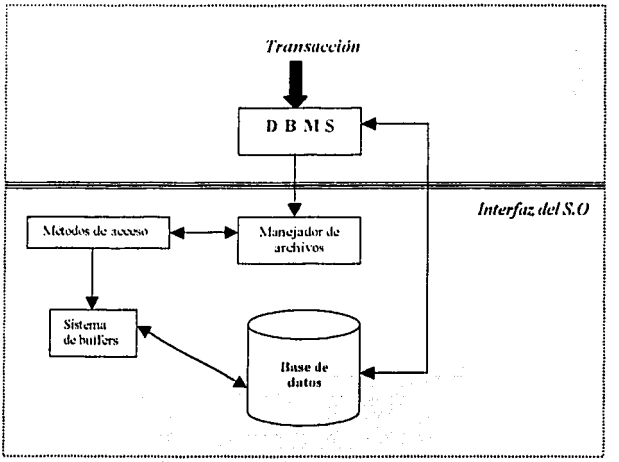


Figura 1.3 Esta figura muestra cada uno de los elementos que interactúan con el DBMS.

El objetivo de presentar esta figura es únicamente mostrar la manera en la que se comunica el DBMS con algunos elementos, si bien la figura muestra muy burdamente el flujo de una transacción en este momento no es importante este aspecto, en capítulos posteriores se explica detalladamente el flujo de una transacción.

La teoría antes mencionada es la base sobre la que están creados los sistemas manejadores de bases de datos relacionales. En los temas siguientes nos enfocamos a la arquitectura y ventajas de un RDBMS con respecto a un DBMS.

Organización de un RDBMS

Aludiendo a la palabra *"organización"* en realidad se hace referencia a la relación existente entre los componentes del RDBMS.

Se distinguen catorce componentes: [VOGO1991]

1. Procesador de entrada/salida
2. Parser
3. Precompilador
4. Control de autorizaciones
5. Verificador de integridad
6. Procesador de actualizaciones
7. Procesador de consultas
8. Optimizador
9. Generador de código ejecutable
10. Manejador del diccionario
11. Manejador de transacciones
12. Manejador de recuperaciones
13. Log book
14. Manejador de datos

1. Procesador de entrada/salida

El procesador de entrada/salida es el encargado de recibir comandos como entrada (específicamente en una consulta), una vez que la consulta recorre la trayectoria, es el mismo procesador quien devuelve una respuesta a la petición hecha anteriormente. Este es el contacto directo e inmediato con el usuario, usualmente es un "*monitor de procesos*", el cuál lleva a cabo la limpieza de las transacciones terminadas, fallidas o interrumpidas; es posible encontrar este componente fuera del RDBMS como parte del sistema operativo.

2. Parser

El parser es el encargado de transformar una consulta en una petición analítica-sintáctica, para después ser analizada por el precompilador. Su funcionamiento es muy similar al parser de un lenguaje de programación de alto nivel que tiene como funciones analizar si una sentencia en SQL está correctamente escrita, para llevar a cabo este análisis se auxilia del diccionario de datos.

3. Precompilador

Su tarea principal es la de analizar una sentencia que contenga comandos DML. Cuando se tiene una sentencia (DML o DDL) anidada es necesario que pase por el precompilador.

4. Control de autorizaciones

Este componente verifica si el usuario que esta realizando la transacción tiene permisos para poder acceder a los datos que está solicitando. Es un tipo de vigilante que protege la seguridad de los datos.

5. Procesador de actualizaciones

Es el encargado de analizar si es posible una actualización y de llevarla a cabo, en esta tarea tiene como ayudante al verificador de integridad, por lo tanto es el responsable de transferir la información correspondiente al nombre de la tabla, de la columna así como el valor nuevo a sustituir al verificador de integridad.

6. Verificador de integridad

Este componente se encarga de vigilar cuando se pretende hacer una actualización que se cumpla la integridad estipulada como "reglas del negocio" que se definieron al inicio de la construcción de la base de datos. Hablando estrictamente del modelo relacional los verificadores de integridad son los constrainsts(condiciones), por ejemplo, cuando se pretende actualizar el salario de un trabajador el constraint (creado previamente) verifica que no se inserte un salario con signo negativo. Los resultados los reporta al procesador de actualizaciones.

7. Procesador de consultas

Si una consulta es formulada en términos del esquema externo, es decir, desde una interfaz, es tarea del procesador de consultas traducir dicha consulta para que el esquema conceptual pueda entenderla y examinarla.

8. Optimizador

Es el encargado de tomar los nombres de tablas y columnas referenciadas por la consulta, buscarlas en el diccionario de datos para verificar su existencia y obtener caminos de accesos almacenados.

Cuando en una consulta se requiere extraer información de más de una tabla, el optimizador realiza permutaciones de los métodos de join, eligiendo así el camino de acceso que minimice el costo para la consulta. Esta solución recibe el nombre de "plan de ejecución", el cual se representa en el lenguaje de especificación de accesos (ASL, *Access Specification Language*) cuya definición queda fuera de los alcances de este trabajo. Solo algunos RDBMS cuentan con un optimizador, entre estos se encuentran Postgress, Sybase y ORACLE.

9. Generador de código ejecutable

El generador de código es un programa que traslada árboles ASL en lenguaje de máquina para ejecutar el plan escogido por el optimizador, para hacer esto utiliza un pequeño número de plantillas de código para representar todos los casos posibles, durante esta fase se reemplaza el árbol generado por el parser en código de maquina ejecutable y sus estructuras de datos asociadas.

10. Manejador de transacciones

La función de este componente es asegurarse que la base de datos esta disponible a cualquier usuario que requiera hacer uso de ella y que cuenta con el permiso. Siendo también el responsable de cuidar la integridad del sistema, debe vigilar que cuando se efectúen muchas consultas simultáneamente no interfieran una con la otra. Así mismo se encarga de que no se pierda información debido a fallas del sistema. Su trabajo consiste en asegurar que todas las consultas se efectúen correctamente. Para llevar a cabo las consultas se requieren de cuatro requerimientos o llamadas también propiedades ACID (*Atomicity, Consistency, Isolation, Durability*).

I. *Atomicidad*: Se refiere a que las transacciones deben ejecutarse o no, pero no pueden quedar en un estado a medias.

II. *Consistencia*: Se refiere a que debemos conocer o sospechar la respuesta a nuestra transacción.

III. *Aislamiento.* Cuando se ejecutan dos transacciones simultáneamente, para resolverlas se deben separar o aislar.

IV. *Durabilidad:* Se refiere a que los efectos de la transacción deben ser permanentes.

11. Manejador de recuperaciones

Este componente se hace presente cuando alguna transacción se interrumpe, el manejador de recuperaciones regresa a la base de datos al estado en el que estaba antes de haber iniciado la transacción para deshacer todos los cambios que la transacción hizo a la base de datos.

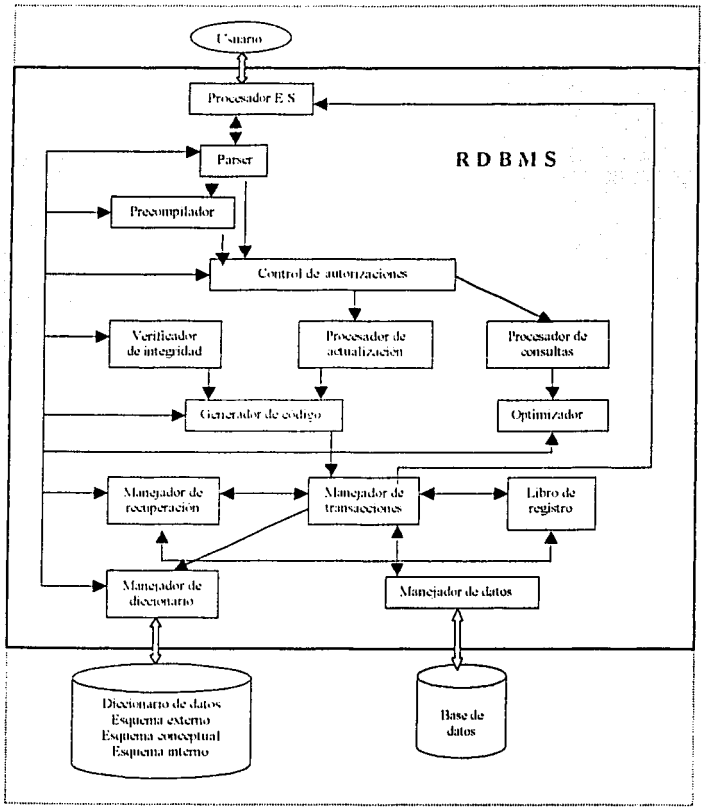


Figura 1.4 Muestra la organización de un RDBMS (organización de sus componentes y las relaciones entre cada uno de ellos).

12. Libro de registro (Log book)

Es el ayudante del manejador de recuperaciones, debido a que es dentro de este componente donde se registran los cambios que se hacen a la base de datos.

13. Manejador del diccionario

Es el encargado de hacer accesible el diccionario de datos a los componentes que lo requieran para llevar a cabo sus funciones específicas en el flujo de una transacción.

14. Manejador de datos

Este componente maneja todo los recursos que están disponibles al RDBMS a nivel hardware, incluyendo dispositivos de almacenamiento.

En la figura 1.4 se distinguen tres elementos que integran la misma: el RDBMS y dos bloques de datos, los cuales son elementos necesarios en el proceso del almacenamiento de los datos.

Uno de los bloques con los que opera el RDBMS es la base de datos, el otro son los esquemas (interno, conceptual y externo) los cuáles están asociados con los tres niveles de la arquitectura del manejador. Estos bloques (esquemas y base de datos) a diferencia del manejador se encuentran almacenados en la memoria secundaria.

Fundamentos de un RDBMS

Las primeras tres características de un RDBMS se encuentran basadas en el álgebra relacional, la cual es uno de los pilares sobre los que se asienta el modelo relacional junto con el cálculo relacional. El resto de las características hacen referencia al funcionamiento del sistema manejador de bases de datos relacional.

- *Operador de proyección*

Mediante este operador es posible extraer información de una base de datos construida desde una perspectiva relacional. Este operador permite hacer referencia y desplegar el contenido de columnas específicas de una tabla específica. Hablando en términos de SQL, cada vez que se incluye el comando SELECT en una sentencia se utiliza el operador algebraico relacional de proyección.

- *Operador de selección*

Este operador permite condicionar la extracción de los datos contenidos en una tabla. Haciendo referencia al Lenguaje Estructurado de Consulta, podemos asegurar que en cada sentencia que se incluya la cláusula WHERE se lleva a cabo una condición específica de la información.

- *Operador de Join*

Este operador también es útil para especificar condiciones en la información que es solicitada a una base de datos mediante un RDBMS. Matemáticamente se define como el resultado de realizar el producto cartesiano de dos o más relaciones el cual es restringido por patrones de búsqueda, tales como igualdad a un valor numérico, a una lista de caracteres, etc. En la cláusula WHERE es donde se especifica la utilización de un operador de join.

- *Definición de datos.*

Cuenta con un lenguaje de definición de datos (DDL, *Data Definition Language*), se utiliza para definir la estructura de la base de datos esto incluye crearla, borrarla o alterarla.

- *Manipulación de datos.*

Cuenta con un lenguaje de manipulación de datos (DML, *Data Manipulation Language*), permite manejar los datos de la base de datos, así se puede insertar, borrar o actualizar renglones en una tabla.

Estos dos grupos de comandos (DDL y DML) forman ahora lo que es el lenguaje estructurado de consulta (SQL), el cuál es el lenguaje estándar para el manejo de las bases de datos relacionales.

- *Seguridad de los datos.*

El RDBMS permite al administrador de la base de datos (DBA, *Data Base Administrator*) proteger los datos de cualquier mal uso que pudieran tener por parte de los usuarios, es por eso que asignan passwords y otorga privilegios sobre la base de datos, así todo usuario cuenta con una lista de permisos tales como conectarse, crear, borrar, actualizar vistas o crear funciones, procedimientos, etc.

- *Integridad de los datos.*

La integridad de los datos se hace válida al implementar en la base de datos las famosas "reglas del negocio".

- *Recuperación de los datos.*

Un RDBMS debe contar con un sistema que permita hacer copias de la información (respaldos) debido a algún fallo en el sistema que pueda ocurrir inesperadamente. Igualmente debe responder a fallos de hardware, errores en el código de las aplicaciones, etc.

- *Concurrencia de los usuarios.*

Esto es atender peticiones de varios o muchos usuarios simultáneamente. Un DBMS debe ser capaz de controlar la concurrencia y atender las peticiones de cada uno de los usuarios finales por individual.

- *Diccionario de datos*

Se conoce la definición formal de un diccionario de datos como "una base de datos de los datos", (meta datos) lo cual implica contar con información referente a la descripción, administración e implantación de la base de datos.

Los aspectos abordados a lo largo de este capítulo son suficientes para entender los siguientes capítulos.

En el siguiente capítulo se aborda la manera en la que se almacenan los datos físicamente en los diferentes dispositivos de almacenamiento.

CAPITULO II

ALMACENAMIENTO FISICO DE LOS DATOS EN UN SISTEMA MANEJADOR DE BASES DE DATOS

Este segundo capítulo aborda todos los aspectos que intervienen en el almacenamiento físico de los datos. Primeramente se hace referencia a la jerarquía que se establece entre los diversos dispositivos de almacenamiento para así continuar con la distinción de las estructuras físicas (pistas, cilindros, etc.) y lógicas (páginas, frames, etc.)en que un disco magnético se divide para almacenar los datos. Teniendo una explicación clara de las características esenciales de los componentes físicos de un disco magnético en los cuáles se lleva a cabo el almacenamiento de los datos, se hace referencia al flujo completo que sigue una petición de información desde los componentes externos (sistema operativo y esquemas de la base de datos) al DBMS hasta los componentes internos (manejador de buffer, administrador de archivo y disco) del mismo. Se concluye este capítulo con un ejemplo práctico del almacenamiento de datos.

Jerarquía de almacenamiento

El primer aspecto a considerar es conocer la jerarquía de almacenamiento que existe entre los diferentes dispositivos de almacenamiento que existen dentro de una computadora.

El almacenamiento se divide por niveles, la diferencia entre cada uno de los niveles además del precio, es la rapidez para acceder a la información almacenada en cada uno de los dispositivos de cada nivel. Así por ejemplo la memoria caché (almacenamiento primario) es más cara que una cinta magnética (almacenamiento terciario), pero por supuesto también es más rápido el acceso a los datos.

Almacenamiento primario: Se le conoce como almacenamiento primario a la memoria caché y a la memoria principal.

Almacenamiento secundario: Dentro de este nivel se encuentran los discos duros y RAID (*Redundant Arrays of Inexpensive or Independent Disks, Matriz redundante de discos económicos o independientes).*

Almacenamiento terciario: Se ubican a las cintas magnéticas, discos ópticos, etc. La siguiente figura muestra la jerarquía antes explicada.

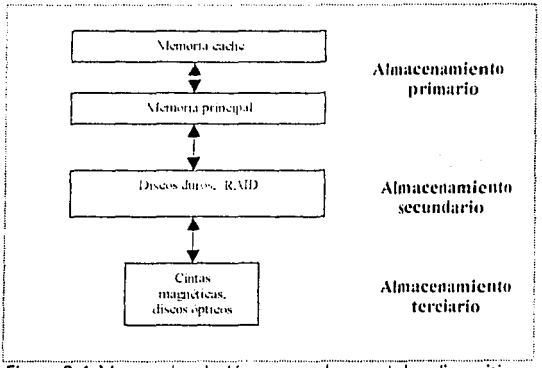


Figura 2.1 Muestra la relación entre cada uno de los dispositivos de almacenamiento y al nivel que pertenecen cada uno de estos.

Características de los diferentes niveles de almacenamiento.

Almacenamiento primario

- **MEMORIA CACHÉ**: Impone al sistema un nivel de traspaso, ya que los programas son enviados desde la memoria principal a la memoria caché antes de ser ejecutados, debido a esto, dichos programas se ejecutan más rápidamente lo que implica un costo más elevado que el que tiene la memoria principal, por tanto, la memoria caché es considerada el medio de almacenamiento más caro que existe en el mercado.
- **MEMORIA PRINCIPAL**: Es el dispositivo donde se almacenan temporalmente tanto los datos como los programas que la CPU está procesando o va a procesar en un determinado momento. Es la encargada de llevar los programas del almacenamiento secundario (discos duros, RAID) a la memoria caché.

Almacenamiento secundario

- **DISCO DURO**: Se usan como dispositivos de almacenamiento de acceso directo en línea, y son los medios que comúnmente se utilizan para almacenar una base de datos. En el siguiente tema se explica detalladamente el almacenamiento de los datos en disco duro.

- **RAID:** Es un dispositivo poco conocido ya que se desarrolló recientemente; es un conjunto de discos que funcionan en paralelo y su mayor ventaja es que reduce el tiempo de entrada/salida al igual que proporciona mayor seguridad a los datos. La corrección de errores se controla mediante ciertas fórmulas y algoritmos matemáticos que permiten reconstruir los datos perdidos en un disco que ha fallado. Los discos utilizados en RAID pueden ser grandes o pequeños dependiendo de la aplicación de base de datos con la que se cuente, así por ejemplo **DSS** (*Decisión Support system, Sistema de toma de decisiones*) funciona mejor usando discos grandes contrariamente **OLTP** (*Online Transaction Processing, Procesamiento de transacciones en línea*) funciona mejor con discos pequeños. Hay varios niveles de RAID pero los principales y de los cuales se derivan los demás son 0, 1, 3 y 5. La capacidad de almacenamiento es de varios terabytes.

Almacenamiento terciario

Los dispositivos considerados dentro del almacenamiento terciario se crearon para transportar una gran cantidad de datos, a diferencia de los dispositivos anteriormente mencionados son en gran escala de menor costo. Los principales dispositivos de almacenamiento terciario son:

- **CINTA MAGNÉTICA:** El almacenamiento en cinta es secuencial, de esta característica se desprende su utilidad: respaldar información contenida en discos duros.
- **DISCOS ÓPTICOS**: Dentro de éstos se encuentran los CD-ROM's que tienen una capacidad de almacenamiento de 600 Mb (Mega bytes) y los DVD ROM's que pueden almacenar entre 1.7 y 17 Gb (Gyga bytes).

Almacenamiento de datos en disco

Enfocamos la atención a los discos duros, que como es bien sabido son el medio de almacenamiento más usado, se necesita una explicación detalladamente de la arquitectura del disco magnético para así posteriormente entender la forma en que los datos son almacenados en este.

El objetivo de hacer énfasis a los discos magnéticos se debe a que generalmente la base de datos completa se respalda en ellos; por tanto, el almacenamiento en disco es la forma principal de almacenamiento con acceso directo.

La unidad física que contiene el disco magnético se llama *"controlador de disco"*. El controlador de disco contiene un paquete de discos que están montados sobre un eje

(Fig. 2.2). Cada disco proporciona dos superficies de grabación en las cuáles se graban los datos y dos cabezas de lectura/escritura que están unidas a un brazo de acceso, una para cada superficie de grabado.

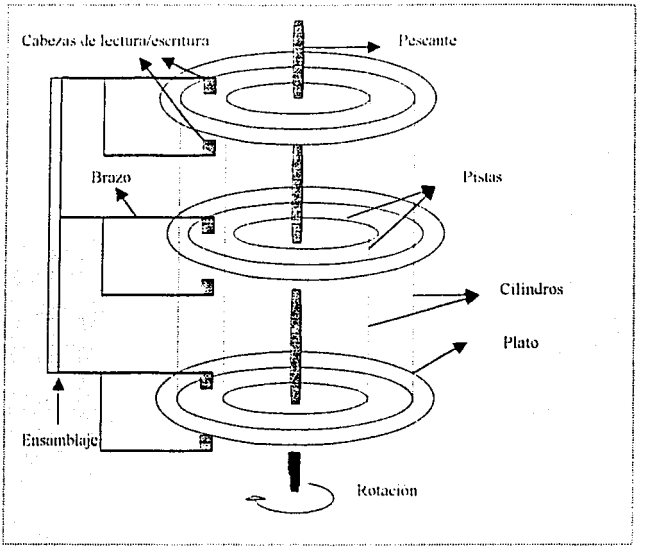


Figura 2.2 Organización de un paquete de tres discos, este paquete cuenta con cuatro superficies de grabado, dos del disco que está en medio y una de cada disco que se encuentran en los extremos, las otras dos superficies se utilizan para protección.

Organización de discos

Un paquete de discos se organiza en **pistas** y **cilindros** para localizar el lugar en donde se encuentran almacenados los datos, éstos se graban en la superficie del disco en anillos, cada anillo representa una pista (*track*). En una superficie hay cientos de pistas y todas almacenan el mismo número de bytes. Cada pista en un disco está dividida en sectores los cuales a su vez forman un bloque. (Fig. 2.3). El número de pistas de un disco puede llegar hasta 800 y la capacidad de cada pista oscila entre 4 y 50 KB.

El número de sectores que forman un bloque es directamente proporcional a la extensión del registro, así el tamaño del sector lo determina el sistema operativo que

se tiene instalado en la computadora, contrariamente el tamaño de un bloque se fija durante la iniciación y no se puede modificar dinámicamente, el tamaño normal se sitúa entre 512 y 4096 bytes.

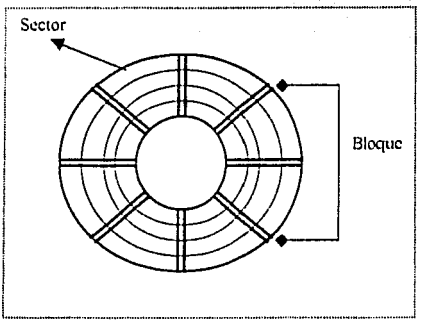


Figura 2.3 Vista de la superficie de un disco mostrando los sectores y bloques.

DEFINICIÓN: Un conjunto de arandelas (pistas) con el mismo diámetro en diferentes superficies constituye un ***cilindro.***

El concepto anteriormente mencionado es muy útil, ya que cualquier posicionamiento de cabezas de lectura/escritura puede ser descrito por la localización del cilindro.

El mecanismo de acceso consiste en un grupo de brazos montados en un ensamblaje (Fig. 2.2) llamado *ensamblaje de acceso*, éste se mueve horizontalmente hacia delante y hacia atrás desde el cilindro exterior al interior.

El paquete de discos gira sobre su eje a una velocidad constante, teniendo así acceso a los datos mediante las cabezas de lectura/escritura que se desplazan entre los cilindros para después ubicarse en el diámetro apropiado y así estar en contacto directo con los datos que estarán en un máximo de una revolución.

El tiempo de acceso a disco tiene tres componentes:

I. ***Tiempo de búsqueda*** (*seek time*): Es el tiempo (en milisegundos) que le lleva al ensamblaje moverse a un cilindro.

II. ***Tiempo de latencia*** (*latency time*): Es el tiempo que emplea en rotar la cabeza de lectura/escritura para iniciar la transferencia de datos.

III. *Tiempo de transferencia* (*transfer time*): Es el tiempo para leer o escribir los datos una vez que la cabeza de lectura/escritura está perfectamente posicionada.

Organización de registros

Los datos almacenados en una base de datos en realidad están contenidos físicamente en los dispositivos de almacenamiento secundarios vistos en el tema anterior (discos duros), la organización de los datos dentro de estos dispositivos se lleva a cabo en archivos. Cada archivo está compuesto de una serie de registro. El tamaño de un registro es el número de bytes que contiene y el número de bytes se determina por el tipo de datos que se almacena (carácter, numérico, etc.).

Un archivo puede estar compuesto por dos tipos de registros:

1) *Archivos de registros de longitud fija.* Todos los registros son del mismo tamaño.
2) *Archivos de registros de longitud variable*: Los registros pueden tener diferente tamaño.

Registros de longitud fija

Características del manejo de registros de longitud fija:

- Dificultad para eliminar registro, debido a que el espacio que ocupa el registro que se va a eliminar debe llenarse con otro registro, o se debe señalar para que sea ignorado.
- Ocupación de más de un bloque, lo que implica hacer más de un acceso a disco para la lectura o escritura de un registro.
- Problemas con el manejo del espacio, al momento de eliminar un registro quedan huecos.

Registros de longitud variable

El manejo de registros de longitud variable es utilizado por los DBMS para el almacenamiento de los datos, se presentan de varias formas:

1. Almacenamiento de varios tipos de registros en un archivo.
2. Tipos de registros que permiten uno o más campos de longitudes variables.
3. Tipos de registros que permiten campos repetidos.

Existen varias técnicas para implementar los registros de longitud variable.

Representación en cadena de bytes

Un método sencillo para implementar los registros de longitud variable es añadir un símbolo especial de fin de registro al final de cada registro, permitiendo así almacenar cada registro como una cadena de bytes consecutivos.

Esta representación presenta dos desventajas:

1. Dificultad para la reutilización del espacio que ocupaba un registro eliminado.
2. Indisponibilidad de crecimiento de los registros, si un registro de longitud variable se hace más largo debe moverse, y si el registro está sujeto, el movimiento resulta costoso. Debido a las desventajas que presenta la representación en cadena de bytes, no se utiliza para almacenar registros de longitud variable.

Representación de longitud fija

Esta técnica permite a todos los registros que están contenidos en un bloque tener la misma longitud, aunque los registros del archivo sean de longitud variable.

En el siguiente capítulo se muestra detalladamente los tipos de organizaciones de archivos.

Antes de profundizar en la explicación de las estructuras de almacenamiento, es necesario conocer la trayectoria que recorre una petición, tema que se aborda a continuación.

Acceso físico a la base de datos

Flujo de una petición

Los elementos que intervienen en el flujo de una petición se pueden agrupar en tres bloques muy generalizados:

1. *DBMS*
2. *Sistema operativo*
3. *Esquemas de la base de datos*

La figura 2.5 muestra los componentes que intervienen en el flujo o trayectoria que recorre una petición[BEMA, 1994].

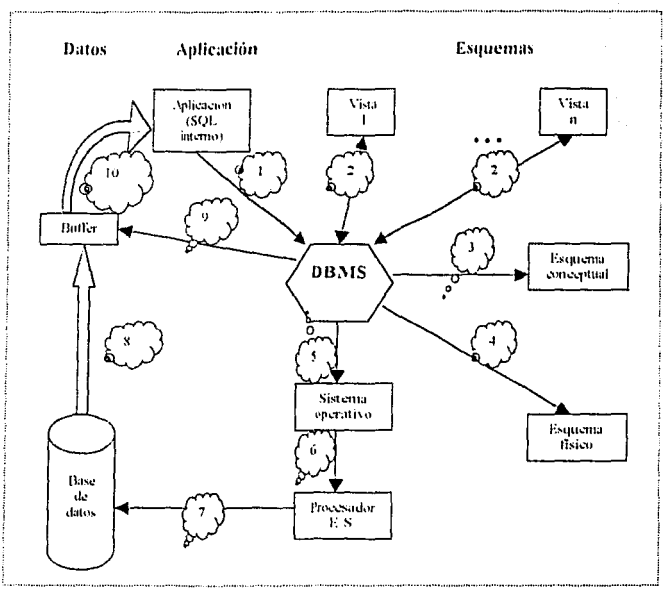


Figura 2.5 Muestra la trayectoria que recorre una petición.

Haciendo referencia a la figura, supóngase que a través de la aplicación se solicita información de la base de datos (consulta), esta consulta que es una declaración de lectura sobre la base de datos se envía al DBMS (paso 1), una vez que llega al DBMS se debe asociar dicha declaración de lectura con un subesquema y verificar la validez de la misma (paso 2), entonces el DBMS mapea el subesquema a su correspondiente esquema conceptual y determina el tipo de archivo lógico que debe ser buscado (paso 3), posteriormente el DBMS consulta el esquema físico para determinar el archivo físico que se requiere para satisfacer la petición e identifica la ubicación de los archivos de la base de datos (paso 4); a partir de este paso y hasta llegar a los últimos, es la parte donde se enfoca esta tesina ya que el propósito general de la misma es el conocer el proceso de almacenamiento físico de los datos que lleva a cabo el DBMS.

Después del paso 4, el DBMS transforma la petición lógica en una petición física que debe ser leída por uno o más archivos; los que sean necesarios. Esta petición es enviada al sistema operativo (paso 5). El sistema operativo analiza la petición de entrada/salida, esto lo hace consultando el esquema físico, en este se conoce el

tamaño y ubicación de los archivos y se lleva a cabo el manejo de los dispositivos de almacenamiento del sistema (paso 6).

Una vez que se conocen todos los elementos necesarios para conocer la ubicación exacta de los datos en la base de datos se procede a extraer la información que satisfará la petición que en un principio se hizo, específicamente es el buffer del DBMS el que selecciona los datos que cumplirán la petición (pasos 7 y 8) y finalmente los datos que son requeridos regresan a la aplicación mediante la cuál fueron solicitados.

Esta explicación es muy general, por tanto en las siguientes líneas se abordan detalladamente los elementos que integran el proceso de manejo del almacenamiento físico de los datos: estructuras de almacenamiento como el manejador de buffer, administrador de archivos y el administrador de disco, ya que son las principales estructuras que intervienen directamente en el almacenamiento físico de los datos.

La siguiente figura muestra la trayectoria que recorre una petición al igual que la figura anterior la diferencia radica en que la figura 2.6 integra a la memoria principal, al administrador de archivos y al administrador de disco.

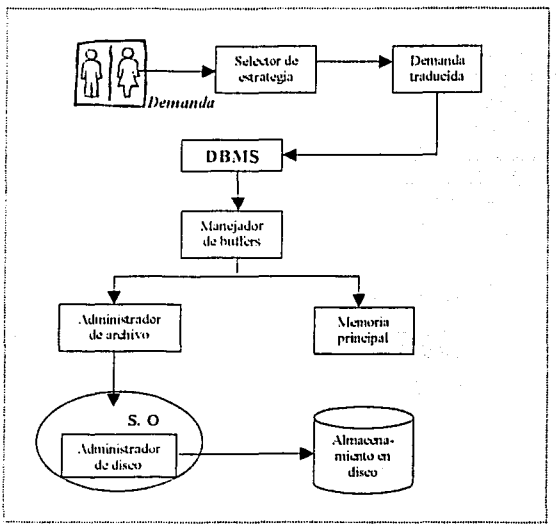


Figura 2.6 Muestra el lugar que ocupan el manejador de buffers, el administrador de archivos y el administrador de disco en la trayectoria que sigue una petición.

Habiendo explicado claramente el flujo de una petición, solo falta explicar los componentes que se integran adicionalmente a la figura 2.6 en comparación con la figura 2.5. Tales como el selector de estrategia, administrador de archivos y administrador de disco.

Una vez que el usuario hace su petición el ***selector de estrategia*** traduce la petición del usuario en una petición ejecutable, en esta figura le llamamos así pero no es sino la asociación de los pasos 2, 3 y 4 que se explicaron anteriormente en la figura 2.5.

Es conveniente aclarar que los datos del usuario se almacenan como una colección de registros. Así por ejemplo una tupla en una tabla puede almacenarse como un registro físico en donde cada valor de los atributos de dicha tupla se almacena en su propio campo.

La petición traducida activa el ***manejador de buffer*** el cual controla el movimiento de datos entre la memoria principal y el almacenamiento en disco, para entender este concepto es necesario decir que un **buffer** es un depósito en el que se guarda algún bloque de datos para ocuparlo posteriormente cuando sea necesario.

DEFINICIÓN: Un **manejador de buffer** (*buffer manager*)es una capa de software que tiene como tarea principal traer los datos que se necesitan desde el disco duro a la memoria principal, este proceso pareciera muy fácil a simple vista sin embargo están implicadas muchas otras cosas como veremos a continuación.

Manejador de buffer

Con la ayuda del administrador de archivos, el manejador de buffer obtiene del disco duro el bloque en el que se encuentran los datos requeridos por el DBMS y escoge la página de la memoria principal en la cuál se va a almacenar dicho bloque. Una página es la unidad lógica mínima en la que se almacena un archivo.

Para entender específicamente lo que hace un manejador de buffer supongamos que una base de datos tiene 1000 páginas, pero la memoria principal solo puede alojar 100, considere que la consulta que el usuario realiza necesita hacer un barrido de todo el archivo; debido a que todos las páginas no pueden ser alojadas en la memoria principal simultáneamente el DBMS debe traer las páginas a la memoria como se requieran, en el proceso, decide que páginas se deben reemplazar para poder alojar a las que faltan, este proceso es conocido como *Política de reemplazo.*

El manejador de buffer por tanto administra el lugar que se encuentra disponible en la memoria principal particionándola en una colección de páginas, las cuáles forman lo que es conocido como *buffer pool* (figura 2.7), dichas páginas son llamadas *frames.*

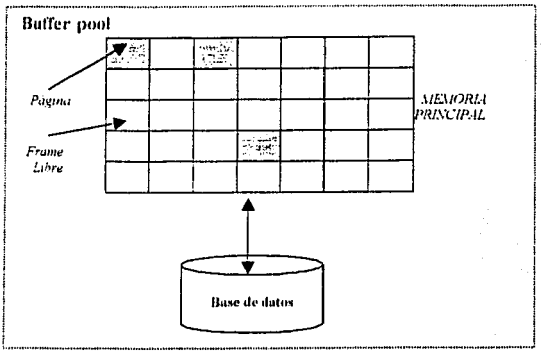


Figura 2.7 Muestra los componentes de un buffer pool (páginas y frames).

El manejador de buffer cuenta con dos variables por cada frame que sirven para conocer el estado del mismo dentro del buffer pool estas son: *pin_count* y *dirty* (sucio), la primera lleva la cuenta del número de veces que una página ha sido solicitada en un frame pero no ha sido reemplazada. La segunda es una variable booleana ya que indica si una página ha sido modificada desde que fue alojada por vez primera en el buffer pool.

Cuando una página es requerida, el manejador de buffer hace lo siguiente:

1.- Checa el buffer pool para ver si la página solicitada está contenida en él, si la página no se encuentra en el buffer pool, el manejador de buffer realiza:

a) Escoge un frame para ser reemplazado (*Política de reemplazo*) e incrementa la variable pin_count.
b) Si la variable dirty bit del frame a ser reemplazado se encuentra encendida, hace una copia de la información que contiene la página en el disco duro.
c) Lee la página que es requerida en el frame reemplazado.

2.- Regresa la dirección de la memoria principal del frame que ha sido reemplazado.

Políticas de reemplazo de páginas

Una de las mejores políticas de reemplazo de páginas es el tan conocido algoritmo **LRU** (***least recently used***) o **"Menos recientemente usado"**. Esta estrategia selecciona para su reemplazo a aquella página que no ha sido utilizada durante el

mayor tiempo, LRU exige que se marque cada página en el momento en que se hace referencia a ella. Esto implica mucho trabajo adicional debido a esto lo que se hace es utilizar estrategias que se aproximen a LRU y que no ocasionen esos grandes costos.

La estrategia LRU se puede realizar con una estructura de listas que contenga una entrada por cada frame de página ocupado. Cada vez que se hace referencia a un frame de página, la entrada correspondiente a esta página se coloca al principio de la lista, y las entradas más antiguas se llevan al final de la lista. Cuando hay que reemplazar una página para dejar espacio a otra entrante, se selecciona la entrada al final de la lista, se libera el frame de página correspondiente, se coloca la página entrante en el frame de página y la entrada correspondiente a ese frame de página se coloca al principio de la lista, porque esa página es ahora la que ha sido utilizada más recientemente.

Como ejemplo práctico de estas políticas de reemplazo tenemos a los RDBMS comerciales Informix y Oracle que usan la política de reemplazo LRU, a continuación se explica la manera en que ORACLE usa esta política de reemplazo.

La política de reemplazo LRU es utilizada por ORACLE para cambiar los datos de la caché de buffers de base de datos* (componente de una instancia ORACLE) apoyándose en una lista de los menos recientemente usados (LRU). Esta lista LRU controla los bloques de datos a los que se tiene acceso y la frecuencia con la que se accede a ellos.

Cuando un usuario hace una petición de información antes de buscar en disco se hace una búsqueda en la caché de buffers de base de datos para ver si los datos requeridos están alojados en ésta. De ser así, mueve los bloques al extremo **MRU** (*Most Reciently Used*) indicando con esto que dichos bloques son los más recientemente usados y por tanto es conveniente tenerle más tiempo en la caché de buffers. En caso contrario, es decir, que no estén los datos solicitados por parte del usuario en la caché de buffers, se deben buscar en los datafiles (capítulo IV).

Cuando los datos deben ser leídos del disco, primeramente se debe buscar en la lista LRU, ORACLE recorre dicha lista para encontrar bloques libres suficientes para almacenar los datos leídos del disco, cuando el RDBMS encuentra bloques libres lee los datos desde el disco colocándolos en el extremo MRU y así indicar que han sido recientemente usados, en caso de no existir bloques disponibles ORACLE escribe bloques sucios (dirty blocks)** en el disco para así conseguir espacio para leer nuevos bloques que irán a la lista LRU [PAGW, 2001].

*La caché de buffers de base de datos se encuentra compuesta por bloques de memoria que tienen el mismo tamaño que los bloques de datos ORACLE (capítulo IV).

**El manejo de los dirty blocks es propio del proceso de background DBWR que son parte de la arquitectura de una instancia ORACLE, escapando así al alcance del presente trabajo.

Una variante de LRU es el reemplazo de páginas por **"Reloj"** (***Clock***). Esta política de reemplazo dispone las páginas en una lista circular, en lugar de en una lista lineal. Un apuntador a la lista se desplaza alrededor de la lista circular hacia la derecha. Cuando el bit de referencia de una página toma el valor 0, el apuntador se mueve al siguiente elemento de la lista (simulando el movimiento de esta página al final de la lista FIFO). SQL Server de Microsoft usa esta política de reemplazo.

Otra política de reemplazo de páginas es: **"Primero en entrar primero en salir"** (**FIFO,** *First Input First Output*). En el reemplazo de páginas de primeras entradas-primeras salidas (FIFO), para cada página se registra el momento en que entró en el almacenamiento primario. Cuando se necesita reemplazar una página, se escoge la que ha permanecido en el almacenamiento durante mayor tiempo. Por intuición, esta estrategia parece razonable, pues esa página ha tenido su oportunidad y es tiempo de darle una oportunidad a otra página. Lamentablemente, es probable que esta estrategia reemplace páginas muy utilizadas, ya que si una página permanece en almacenamiento primario durante mucho tiempo puede deberse a que está en uso constante.

Ya se ha descrito ampliamente la capacidad del manejador de buffer en el flujo de una petición, pero aún falta determinar el papel que desempeñan el administrador de archivos y el administrador de disco en el manejo del almacenamiento de los archivos; aunque el administrador de disco sea parte del sistema operativo es necesario mencionarlo ya que juega un papel muy importante en el manejo de los datos.

Administrador de archivos y administrador de disco

Teniendo ya en la memoria principal las páginas necesarias para satisfacer la demanda hecha por el usuario; específicamente lo que sucede entre el administrador de archivos y el administrador de disco es lo siguiente:

1. El DBMS decide que archivo es requerido y pregunta al *"administrador de archivos"* si puede hacer uso de dicho archivo.

2. El administrador de archivos convierte esta petición en la dirección de una página (unidad lógica mínima en la que se almacena un archivo), decide que página contiene el archivo solicitado por el DBMS y pregunta al *"administrador del disco"* si puede hacer uso de dicha página.

3. El administrador del disco determina la ubicación física de la página y pasa esta información al administrador de archivos.

4. Finalmente el administrador de archivos extrae el archivo pedido de la página y lo pasa al DBMS.

Esta explicación se visualiza claramente en la figura 2.8.

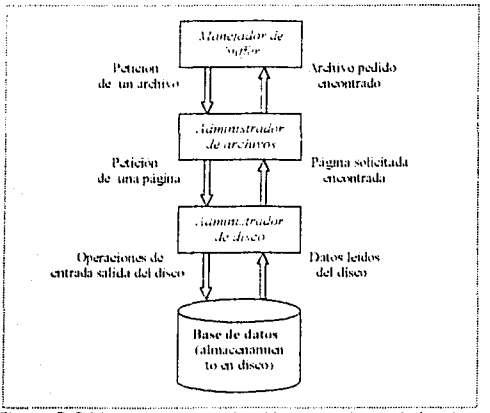


Figura 2.8 Muestra la comunicación entre el manejador de buffer y los administradores de archivos y de disco.

Administrador de archivos (File manager)

Se encarga de manejar el almacenamiento de los archivos, esto es, no se ocupa de conocer la dirección física de la ubicación de estos, sino controlar cuantos archivos están contenidos en una página. Esto lo hace mediante la identificación de cada uno de los archivos por medio de la asignación de un nombre o de un número único (id) a cada uno de los archivos que forman una página.

Dentro de las tareas que lleva a cabo el administrador de archivos están: la creación, eliminación, actualización y reubicación de un archivo.

Se ha explicado anteriormente el almacenamiento de los datos en una página: la división de la memoria principal y todo lo referente a ella, un ejemplo real es una buena herramienta para el entendimiento preciso de este tema. Para comprender como es que se almacenan los renglones en una tabla se presenta el siguiente ejemplo.

Administrador de disco (Disk manager)

El administrador de disco forma parte del sistema operativo y es el responsable de las operaciones físicas de entrada/salida. Es el encargado por tanto de conocer y controlar la ubicación de cada una de las páginas que están contenidas en un disco, este proceso lo lleva a cabo mediante la asignación de un número único (identificador, *id*)a cada una de éstas, por ejemplo cuando el administrador de archivos solicita una página, el administrador de disco sabe la ubicación exacta (número) de dicha página.

Dado que solo se conoce el número único de la página y se requiere la dirección física de la misma dentro del disco, es tarea también del administrador de disco hacer la conversión entre el número de identificación de la página y la dirección física para así poder leer los datos contenidos en dicha página.

Cuando se actualiza un archivo el administrador de disco ubica la página donde está contenido ese archivo (una página puede contener varios archivos), actualiza éste y reemplaza la página. Una de las principales tareas del manejador de disco es no mostrar al manejador de archivos los detalles de la ubicación física de los archivos.

Almacenamiento de los datos en una tabla

Supóngase una tabla llamada ALUMNOS que cuenta con cuatro atributos: matrícula del alumno(id_alumno), nombre (nombre_a), apellido (apellido_a) y clave de la carrera en la cuál está inscrito(clave_carrera). La figura 2.9 muestra el contenido de la tabla.

Tabla: **ALUMNOS**

ID_ALUMNO	NOMBRE_A	APELLIDO_A	CLAVE_CARRERA
001	Juan	Perez	2
002	Ana	Garcia	7
003	Irene	Gomez	9
004	Adrian	Coria	3
005	Erika	Trejo	7

Nota: Para poder identificar el nombre de la carrera debemos crear una tabla llamada CARRERA que contenga el nombre de la carrera correspondiente a cada clave de la misma.

Figura 2.9 Muestra los datos contenidos en la tabla ALUMNOS.

Considere la secuencia de eventos que suceden:

1.- Los cinco renglones de alumnos (001-005) son insertados y almacenados en alguna página 1 (Puede haber más de un archivo en una página). La siguiente figura muestra los renglones almacenados en la página.

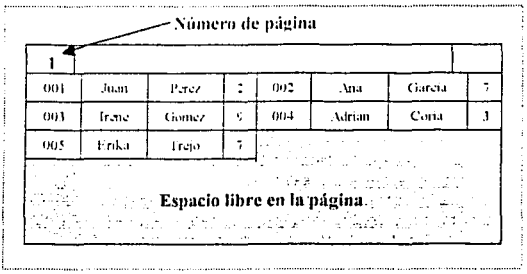


Figura 2.10 Representacion física del almacenamiento de los renglones en una página

2.- Suponga que el RDBMS inserta un nuevo renglón, al alumno con matrícula 008 de nombre Braulio de apellido Jiménez que se encuentra inscrito en la carrera 2. El manejador de archivo almacena este renglón después del alumno con matrícula 005.

3.- El RDBMS elimina el renglón del alumno 003 y del alumno 004. El manejador de archivo borra al alumno 003 de la página 1 y recorre los renglones restantes.

El aspecto a destacar en este ejemplo es la función que realiza el manejador de archivo, pues es éste el que se encarga de mantener en secuencia a los renglones almacenadas dentro de la página.

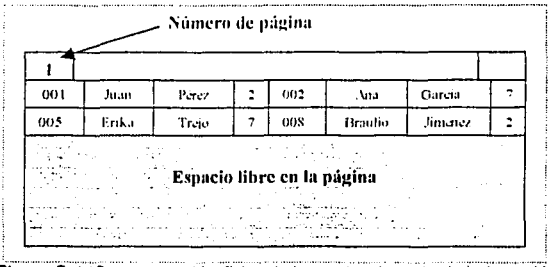


Figura 2.11 Representación física de las tuplas después de la inserción del alumno 008 y la eliminación de los alumnos 003 y 004.

Como anteriormente se explicó los archivos al igual que las páginas se identifican por un número único (id). El id que identifica a un archivo se forma de dos partes: el número de la página que contiene el archivo y el número que identifica al archivo, la figura 2.12 muestra esta implementación.

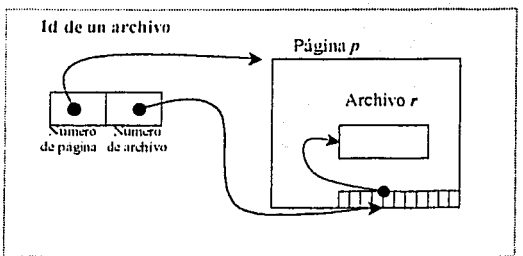


Figura 2.12 Muestra los elementos que construyen el ID de un archivo.

A lo largo de este capítulo se mostró el almacenamiento físico de los archivos en disco, sin embargo para llevar a cabo este proceso se requiere organizar los archivos. Este punto es de suma importancia ya que la buena organización de los archivos incrementa la rapidez en el acceso a los datos.

El capítulo III aborda ampliamente la organización de archivos y la manera en la que se accesa a los datos.

CAPITULO III

ORGANIZACIÓN DE ARCHIVOS Y ACCESO A LOS DATOS

Este capítulo presenta la manera en la que los datos se organiza mediante registros, hace referencia a la organización de los archivos que lleva a cabo el sistema operativo (secuencial, secuencial-indexada, relativa, directa) y también presenta la organización de archivos de alto nivel utilizados por los DBMS comerciales (inversa, multilista, multianillo). Por último aborda una herramienta sumamente eficaz para acelerar el proceso de acceso a los datos: los índices, mencionando los aspectos más importantes de esta estructura.

Organización de archivos y técnicas de direccionamiento

Se ha presentado el almacenamiento físico de los datos en los diferentes dispositivos que están diseñadas para ello, ahora se enfoca la atención a la manera en la que las páginas son almacenadas en disco así como la manera en que éstas son usadas para almacenar y organizar los registros que forman un archivo. Estos aspectos a considerar son conocidos como técnicas de organización de archivos y/o técnicas de direccionamiento de los datos, su uso más relevante consiste en facilitar el almacenamiento y las operaciones de E/S. A continuación se abordan las dos diferentes formas en que los archivos pueden organizarse:

- ***Organización de archivos a nivel Sistema Operativo***
- ***Organización de archivos de alto nivel para sistemas manejadores de bases de datos (DBMS).***

La diferencia más perceptible entre estas dos organizaciones radica en las claves que son asignadas a cada uno de los registros, así por ejemplo en la primera organización de archivos a cada uno de los registros se les asocian única y exclusivamente una clave primaria o identificador contrariamente en la organización de archivos de alto nivel los registros tienen asociadas más de una clave, teniendo una clave primaria, una clave secundaria, etc.

Los tipos de búsqueda para la recuperación de información están basados en tres diferentes tipos de claves:

I. Claves primarias
II. Claves secundarias
III. Claves múltiples

Las consultas en una base de datos muy raramente se basan en los valores de una clave primaria a diferencia de las consultas de claves múltiples que son las más frecuentemente usadas ya que las condiciones de búsqueda no se limitan a un solo valor de la clave principal.

Organización de archivos en el sistema operativo

Cuando se hace referencia al término "*organización de archivos*" en realidad se refiere al método que se usa para vincular y organizar los datos para así poder almacenarlos, procesarlos y recuperarlos.

Primeramente debemos identificar la manera en que los registros se vinculan o enlazan con otros registros, esto lo hacen a través de ***apuntadores ó punteros (pointers).***

DEFINICIÓN: Un **apuntador** es un campo de registro que indica donde se almacena otro registro. Supongamos que tenemos dos registros A y B, y que están organizados de manera secuencial, una representación física de este concepto abstracto sería la siguiente:

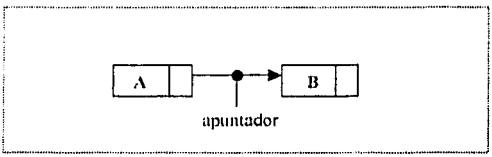


Figura 3.1 Muestra la manera en la que dos registros A y B están vinculados a través de un apuntador (flecha).

Un apuntador puede representar tres cosas diferentes:

1. La *dirección de máquina* del registro al cuál señala.
2. Una *dirección relativa* del registro que señala.

3. Una *dirección simbólica* o *identificador de registro*, que debe ser convertido en dirección de archivo por medio de una técnica de direccionamiento.

Existen diversas formas en que un método de acceso puede organizar los datos en un dispositivo de almacenamiento.

Debido a que la memoria física se encuentra organizada de manera lineal, y cada frame de memoria se debe referenciar por medio de una dirección única, es necesario auxiliarnos de las técnicas de direccionamiento (exploración de un archivo, búsqueda por bloque, búsqueda binaria, etc.) para traducir una petición de recuperación de algún elemento de la estructura lógica a la dirección donde está almacenado físicamente.

Los tipos principales de organización de archivos son [MAJA,1977]:

1. Organización directa
2. Organización secuencial
3. Organización relativa
4. Organización secuencial indexada

Organización directa

En la organización directa el sistema operativo lleva a cabo una distribución automática de manera rutinaria para así determinar las direcciones de máquina de los registros que requieren ser almacenados.

Los registros en un archivo de organización directa se almacenan en bloques sin ninguna secuencia, es decir, de manera muy similar al almacenamiento de los archivos que están organizados relativamente.

Los archivos organizados directamente son útiles para aplicaciones que requieren de un acceso directo rápido.

Organización secuencial

La organización secuencial de un archivo es aquella en la que los registros se almacenan de manera adyacente (uno seguido de tras de otro). Así dichos registros se ordenan ascendentemente o viceversa según la clave (primaria), ésta es uno o varios campos de longitud fija que ocupa la misma posición en todos los registros, por ejemplo la clave que identifica un registro de vuelo en las aerolíneas puede estar

compuesta de dos campos: el primer campo que contiene el número de vuelo y el segundo campo que contiene la fecha del mismo.

La mayor desventaja de esta organización se presenta cuando se necesita acceder a un registro específico, ya que se debe hacer un barrido de todos los archivos desde el primero hasta encontrar el que se necesita, empleando mucho tiempo para encontrar un registro. Este método es por tanto lento para el procesamiento de datos y solo se emplea en las operaciones de **procesamiento por lotes** basadas en el empleo de archivos secuenciales, un ejemplo muy común de esto son las cintas magnéticas que es la forma más económica de almacenar datos.

Organización relativa

La organización relativa de los archivos consiste en la organización de una colección de registros de longitud fija almacenados secuencialmente en un dispositivo de almacenamiento de acceso directo (*DASD, direct-access storage device*), dentro de estos dispositivos se encuentran los discos y tambores [TSHA, 1990].

Se identifica a cada uno de los registros de un archivo organizados relativamente mediante un número entero. Éste número es conocido como **identificador de registro** (llamado también clave primaria) y refleja la posición del registro con respecto al primer registro del archivo además de que a través de este número se accesa (aleatoriamente) a un registro específico. Es muy importante tener presente que el acceso a registros se hace aleatoriamente pues más adelante se hace referencia a este concepto para entender otros.

Una de las dificultades para accesar a un registro de un archivo organizado relativamente es el conocimiento de la dirección relativa del registro y la nulidad de una relación entre la dirección relativa del registro y su clave primaria. Para resolver esta situación se han desarrollado técnicas que tienen por objetivo asociar una clave primaria con su dirección relativa.

Existen varios técnicas para convertir el identificador de un registro o clave primaria en su dirección relativa, las podemos ubicar en dos bloques:

- Direccionamiento clave-igual-dirección.
- Desmenuzamiento ó Distribución Segmentada(*HASHING*).

Direccionamiento clave-igual-dirección

La manera más simple de convertir un identificador de registro en la dirección del mismo registro consiste en agregar a la transacción de entrada la dirección de máquina

del registro en cuestión. Un ejemplo real de la utilización de esta técnica es en las aplicaciones bancarias en las cuáles los números de cuenta de los clientes se modificaron de modo que dicho número de cuenta o parte de él coincidiera con las correspondientes direcciones de los registros. La dirección puede ser igual a la clave primaria o se deriva fácilmente de ésta.

En ocasiones no es posible resolver este problema de manera tan sencilla siendo necesario incluir en la transaccion de entrada un número de referencia de máquina.

De esta técnica tan sencilla se derivan muchas que recurren a tablas para la conversión que se debe hacer o a la asociación de números extras para obtener una dirección relativa a partir de una clave primaria.

Desmenuzamiento o Distribución segmentada (HASHING)

Esta técnica es considerada por muchos la mejor técnica para el cálculo de direcciones relativas, surgió a mediados de la década de los 50's y desde entonces se usa. El desmenuzamiento consiste en convertir a la clave primaria en un número casi aleatorio que posteriormente servirá para determinar la dirección relativa del registro.

La técnica se vuelve más económica cuando se almacenan los registros de un archivo en un área especifica llamada **Cubo**. Un cubo puede contener tantos registros como se quiera pero hay que considerar que a medida que se tienen más cubos, la proporción de desbordes será menor, sin embargo será mayor el espacio de memoria principal necesario para copiar un cubo y mayor el tiempo necesario para barrerlo cuando se busca un registro.

El algoritmo general consta de tres pasos:

1. Si la clave no es numérica, hay que convertirla en numérica. Por ejemplo si tenemos una clave alfabética es conveniente asignarle más de dos dígitos.

2. Se convierten las claves primarias en una lista de números, los cuáles como veremos en las líneas posteriores se pueden sacar de maneras diversas.

3. Los números obtenidos en el paso 2 se multiplican por una constante y el número resultante de esta multiplicación será la dirección buscada.

La técnica HASHING tiene numerosas variantes que siguen los algoritmos descrito anteriormente la diferencia entre ellos es la manera de obtener los números al azar que se mencionan en el paso 2 y la constante requerida por el paso 3 la siguiente lista proporciona algunas variantes:

- Método del centro de los cuadrados
- Método de la división
- Método de desplazamiento
- Método de plegado
- Método de análisis de dígitos
- Método de conversión de raíz
- Método de Lin
- Método de división polinómica

Organización secuencial indexada

Los archivos organizados bajo éste criterio si bien están organizados secuencialmente, es posible acceder directamente a ellos sin tener que hacer un barrido de todo los archivos para localizar uno en particular como se hace en la organización puramente secuencial.

Los archivos secuenciales indexados admiten dos tipos de procesamiento:

1. *Procesamiento secuencial*, en el cuál los registros se toman uno por uno siguiendo un orden.
2. *Procesamiento al azar*, en el cuál los registros se toman de manera aleatoria sin seguir un orden.

Los registros en un archivo secuencial indexado se agrupan en bloques de longitud fija, estos registros se encuentran dentro del archivo de manera secuencial en orden ascendente de acuerdo a la clave primaria de cada uno de ellos.

El procesamiento aleatorio se hace a través de la construcción de un **índice**, para el acceso a un registro a partir de la clave primaria se busca el índice para así encontrar la dirección del bloque en el que se encuentra el registro que se quiere acceder. Este medio facilita en gran manera la localización de los registros.

Debido al papel tan importante que desempeñan los índices en el acceso a los datos, posteriormente se aborda el tema de los índices más profundamente.

Organización de archivos de alto nivel para DBMS

Uno de las cuestiones principales que propició el desarrollo de sofisticados sistemas manejadores de bases de datos fue la necesidad de contar con un software que permitiera efectuar consultas sobre claves múltiples.

Un DBMS altamente desarrollado debe facilitar el establecimiento de organizaciones de archivos de alto nivel para consultas de claves múltiples, debido a esto dicha organización de archivos es también conocida como "**organización de archivos secundaria**".

La figura 3.2 muestra el flujo que siguen los datos en una consulta con archivos de más de un identificador.

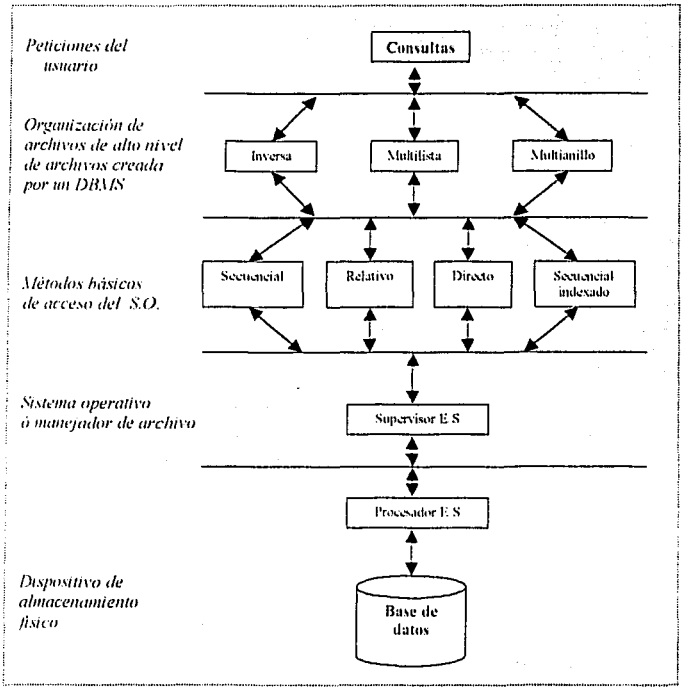


Figura 3.2 Muestra el flujo de datos para consulta de más de una clave.

Un DBMS puede contar con sus propios métodos de acceso para crear bases de datos con una organización de archivos de alto nivel sin embargo puede utilizar los métodos básicos de acceso propios del sistema operativo o del sistema de E/S para manejar los datos.

Las diferentes organizaciones de archivos de alto nivel usadas más frecuentemente por un DBMS son:

1. Organización multilista de archivos
2. Organización multianillo de archivos
3. Organización inversa de archivos

Organización multilista de archivos.

Actualmente muchos DBMS se auxilian de diferentes organizaciones de archivos para vincular los registros que administran, una de éstas organizaciones es la *"organización multilista de archivos"*, que tiene como elemento básico a las listas.

DEFINICIÓN: Una **lista** es una estructura de datos lineal, es decir, es un conjunto de registros distribuidos en archivos vinculados por medio de apuntadores.

Una estructura multilista o también llamada lista con enlaces múltiples consta de diversas listas (cadenas) enlazadas entre sí. Un ejemplo de una serie de registros organizados en una lista o cadena se aprecia en la figura 3.3.

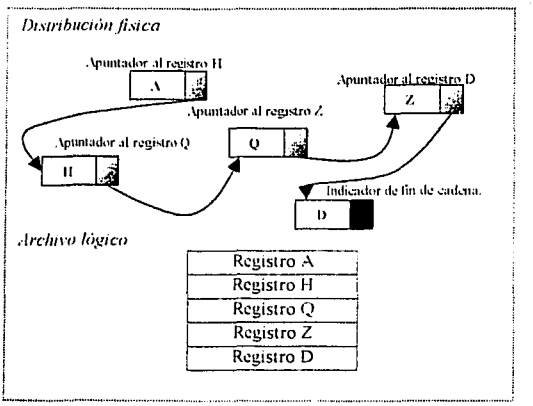


Figura 3.3 Muestra la forma física y lógica como se representa una lista. NOTA: El fin de cadena se indica con el color negro.

Cada registro en una lista tiene un apartado que contiene la dirección del siguiente registro de la secuencia lógica.

Cuando se emplean listas para la organización de los archivos es necesario indicar el inicio y el final de dicha lista. El inicio de la lista se localiza por medio de cualquiera de los métodos de direccionamiento y el fin de la misma se indica mediante un conteo de los registros o bien mediante un indicador de fin de lista insertado en el último registro.

Las listas o cadenas además de enlazar registros tienen otra utilidad, sirven para vincular:

- bloques de la memoria principal,
- bloques de control en los sistemas operativos y
- pilas en la compilación

Inserción y eliminación de registros en una lista.

Cuando se agrega un registro en una lista se puede insertar en cualquier espacio libre del archivo, pero dado que la secuencia de los registros se ve afectada se debe ajustar, esto es, cuando se inserta al inicio de una lista solo el identificador de inicio se corre al nuevo registro que se insertó y se liga con el registro que anteriormente era el primero en la lista; cuando se inserta al final sucede algo similar solo que el que se recorre es el identificador de fin de lista. Si se inserta en medio de una lista se debe igualmente actualizar los apuntadores, esto se aprecia en la figura 3.4.

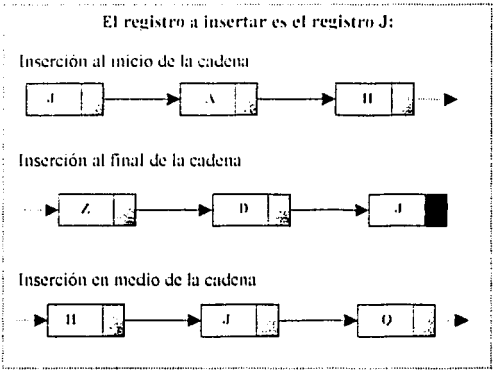


Figura 3.4 Muestra las tres maneras en que se puede insertar un registro.

La eliminación o borrado de un registro se realiza cuando éste queda totalmente desenlazado de algún otro registro, por tanto primeramente debe reubicarse la dirección del apuntador de los registros anterior y posterior al registro que se pretende borrar para así una vez reestructurada la lista el registro a eliminar se quede sin vínculo alguno y poder realizar la eliminación de dicho registro. La figura 3.4 ejemplifica una eliminación, que al igual que una inserción puede ser por cualquier extremo (inicio ó fin) o en medio de la lista.

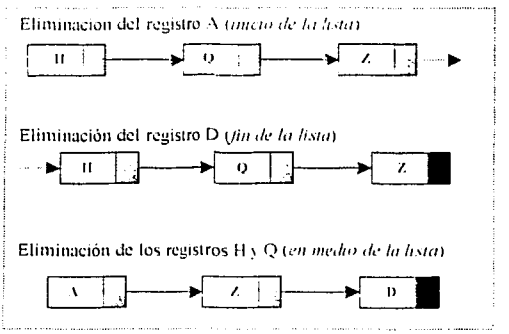


Figura 3.5 Muestra una lista reestructurada después de la eliminación de un registro (inicio, fin o en medio).

Una lista puede estar sola o puede estar intersectada con otra lista (estructura multilista, listas enlazadas) mediante algún registro que tengan en común ambas, para este caso es diferente la eliminación de registros ya que posiblemente se requiera la eliminación del registro que pertenece simultáneamente a las dos listas pero únicamente se necesita eliminar dicho registro de una lista. La solución a este conflicto se resuelve haciendo que el registro en cuestión tenga tres posiciones de *bit de eliminación* (figura 3.6), es decir cada registro de una lista intersectada debe contar con un bit de eliminación por cada lista o cadena a la que pertenezca.

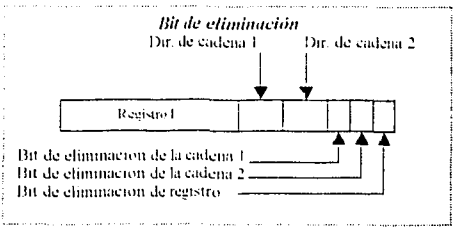


Figura 3.6 Muestra el bit de eliminación en un registro que pertenece a dos listas.

Consultas en una estructura multilista

Una de las desventajas que se presenta cuando se lleva a cabo la implementación de una organización mutilista, es la no-resolución de consultas muy complejas debido a que esta organización es ineficiente para consultas que involucran más de una clave, es decir, cuando utilizan los operadores **AND** y **OR**.

Cuando se involucra el operador AND en una consulta no hay problema alguno para satisfacer dicha consulta, el problema se presenta cuando se involucra el operador OR, pues recordando el álgebra relacional en la que se basan las bases de datos relacionales cuando se efectúa esta operación se presentan duplicidad de resultados; sin embargo para esta organización de archivos existen dos estrategias que minimizan las repeticiones de datos: la primera es la construcción de una *tabla interna de referencia* y la segunda es *una travesía paralela de multilistas.*

Dos variantes con respecto a la estructura básica de multilista son la ***estructura multilista de longitud controlada*** y ***la estructura multilista celular.***

En un archivo *multilista de longitud controlada,* una longitud máxima es impuesta a las listas ligadas de registros de datos. Si algún valor de clave secundaria lo poseen más registros de datos de lo que permite la longitud, entonces el valor de la clave aparecerá más de una vez en el índice y podrá haber más de una lista ligada de registros de datos con ese valor.

Las *estructuras multilistas celulares* son una alternativa a la búsqueda secuencial en listas enlazadas que son muy largas, pues en una organización multilista celular las listas de grandes dimensiones se dividen en segmentos evitando así retrasos prolongados ocasionados por el tiempo de búsqueda que hace la cabeza de lectura/escritura del disco duro cuando accesa a los datos ubicados en zonas físicas (pistas, cilindro o disco) diferentes.

En un archivo multilista celular, las estructuras de la lista están determinadas principalmente por las características del almacenamiento. Por ejemplo, una célula puede ser definida como un cilindro, como una pista, o como una página. A una lista ligada no se le permite cruzar los límites de la célula. Si hay registros en tres cilindros que tienen el mismo valor de clave secundaria, entonces habrá tres entradas de índice y tres listas ligadas para ese valor. Esta variante de multilista puede ser útil en la reducción de movimientos de los brazos lectores del disco y de accesos a la E/S.

Organización multianillo de archivos

La segunda organización de archivos en la que se basan actualmente muchos DBMS es la *"organización multianillo"*, basada en las estructuras de datos conocidas como anillos. Esta organización se sustenta en listas, pues un anillo no es más que una lista en la que se vincula el fin de la lista con el inicio de esta.

DEFINICIÓN: Un **anillo** es una lista o cadena circular en la que el último registro se encadena al primero.

Por esta definición podemos relacionar a los anillos con las estructuras de datos conocidas como colas circulares (término en ingles: *circular queue*).

La siguiente figura muestra un anillo.

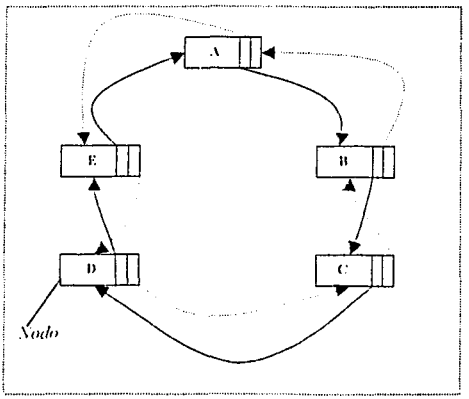


Figura 3.7 Muestra una anillo bidireccional.

Organización inversa de archivos

La tercera organización de clave múltiple a tratar es la organización inversa de archivos la cuál consiste en asociar un índice de claves inversas con los registros de un archivo.

Esta organización inversa de archivos se ha utilizado como base para las estructuras físicas de bases de datos en sistemas manejadores de bases de datos comerciales tales como DB2 y Oracle. Estos sistemas fueron diseñados para proporcionar un acceso rápido a los registros mediante diferentes claves de inversión.

Se dice que los archivos se encuentran organizados inversamente cuando se crea un índice para un archivo directo o secuencial indexado basado en los valores de los campos de la clave-secundaria. Esta acción se asemeja a la creación de un índice en un archivo secuencial indexado en el que cada valor de la clave primaria se asocia con la dirección del registro, pero difiere en la acción de que una clave secundaria puede estar asociada a más de una dirección de registro.

Por cada clave secundaria que exista se debe crear un índice y la colección de los diversos índices que se creen en una organización inversa forman un directorio.

La inversión de la organización de archivos se mide por grados o niveles y se expresa a través del número de campos y la extensión de los valores de los mismos invertidos. Así por ejemplo, cuando se hace referencia a una inversión de un 0% significa que no se invirtió ningún campo clave en el registro, cuando se tiene una inversión de un 25% por ejemplo en un registro que cuentan con 4 campos claves, significa que solo un campo clave se encuentra invertido, etc.

Para poder implementar una organización inversa de archivos se necesita:

A) Espacio de almacenamiento extra para índices.
B) Mantenimiento de los índices cuando se efectúan las operaciones de inserción, borrado o actualización de archivo.

En ocasiones llega a suceder que un archivo índice puede ser tan grande o más como un archivo de datos teniendo dicho índice un grado bajo de inversión, por esta razón no deben crearse índices para campos de datos que tengan posibilidad de uso en condiciones de búsqueda.

Una gran mayoría de DBMS permite al usuario escoger campos clave-secundaria para la inversión. Antes de invertir un campo se debe analizar la eficiencia en el acceso y el costo extra de crear y mantener los índices.

Una vez creados los índices inversos en un archivo de datos, se pueden efectuar consultas sobre claves múltiples a través de sus valores de clave inversa. En consultas sobre claves múltiples, distintas condiciones son conjuntadas mediante el uso de operadores relacionales, si las condiciones de búsqueda se unen con un OR en lugar de AND, el resultado final, será la unión de todos los registros que satisfagan una o ambas condiciones.

Las organizaciones multilista, multianillo e inversa, son las estructuras de archivos fundamentales en la mayoría de los DBMS actuales. Ninguna de las organizaciones puede considerarse mejor con respecto a las demás.

Existen un DBMS particular para el manejo de cada organización de archivos, en general IMS usa la organización multinanillo, los sistemas CODASYL la organización multilista y los RDBMS utilizan la organización inversa de archivos.

Ya hemos visto como es el almacenamiento físico de los datos, las organizaciones de los archivos y las diversas maneras que existen para acceder a los datos, ahora nos enfocaremos a la indexación, la cual es una herramienta que acelera el proceso de acceso a la información contenida en una base de datos.

Indexación

Los índices desempeñan un papel muy importante en el proceso de acceso a los datos que se encuentran almacenados en una base de datos, dado que actualmente una de las características que se buscan en los RDBMS es la rapidez en el acceso a los mismos.

Los índices se crean sobre los archivos y son una estructura auxiliar diseñada para incrementar la rapidez en las operaciones que no son soportadas por las diversas organizaciones de archivos que mencionamos anteriormente (multilista, multianillo e inversa), por esto dedicamos una sección especial a los índices, pues aunque se han mencionado en líneas anteriores son una estructura muy compleja que requiere de un análisis profundo.

DEFINICIÓN: Un **índice** es una estructura que se utiliza para acelerar el acceso a los registros.

La anterior es una definición formal que encontramos en los libros, procesando esta definición podemos decir que un índice es una tabla de apuntadores sobre la que se realizan operaciones de búsqueda.

Los índices tienen diferentes salidas o funciones, es decir un índice arroja diferentes valores que representan lo siguiente:

1. ***Dirección de registros***, un índice provee la dirección máquina del registro que se busca.

2. ***Dirección de registro relativa***, en ocasiones el índice provee una dirección relativa más bien que una dirección absoluta, en estos casos se pueden mover bloques de registros sin necesidad de actualizar todas las entradas del índice.

3. ***Dirección de registro simbólica***, en esta situación el índice provee el identificador que todo registro posee, los índices secundarios que veremos más adelante, son los que frecuentemente arrojan este tipo de salida. El

direccionamiento simbólico de los índices si bien alarga el tiempo de respuesta, facilita el mantenimiento de los apuntadores.

4. ***Localización de cubos,*** algunos índices no señalan registros, sino localidades (llamadas cubo) en las que se alojan los registros, una localidad puede ser una pista o un área de tamaño adecuado en cualquier técnica de indización. El término *resolución* se aplica para hacer referencia al número de registros que se encuentran contenidos en un cubo, y un índice resuelto por cubo provee una dirección de un cubo y un índice resuelto por registro provee registros individuales.

5. ***Dirección de lista,*** en una organización de archivos multilista el índice provee la dirección de la cabeza de la lista.

6. ***Valores de atributo,*** algunos índices secundarios proveen valores de atributo, un valor de atributo provisto por un índice secundario admite la conversión en una dirección de registro de datos siempre y cuando sea sometido a operaciones adicionales de indización.

Un índice debe proveer lo siguiente:

- El estado de un índice debe de reflejar en todo momento el contenido del archivo al que hace referencia.
- El funcionamiento de las operaciones sobre índices asegura que éstos se actualizan en memoria.
- Estas actualizaciones también se deben de reflejar en memoria secundaria, cuando el archivo se cierra.
- Si esta actualización no se produjera, por un fallo del sistema, el archivo índice no reflejaría el estado real del archivo y por tanto debería de reestablecerse.
- Para detectar esta posibilidad, en el registro de cabecera del archivo índice se introduce una bandera que indique esta situación.

Los índices se pueden organizar de diversas formas, las más comunes son:

I. **Índices de un solo nivel**
II. **Índices multinivel**
III. **Índices basados en árboles (trees)**

El índice debe estar organizado en función de alguno de los campos de los registros de datos. Se pueden tener tantos índices como se quiera variando la clave (o campo) que se emplee.

Un índice está formado por registros (entradas) que contienen:

- **Campo de indexación** (clave de organización)
- **Apuntador(es)** al archivo de datos, en concreto al registro que corresponda.

Los índices se pueden clasificar en dos tipos, según cada entrada señale a la dirección de un registro del archivo de datos (*índice total o denso*), o bien apunte a un bloque (sector) de disco del archivo de datos que debe estar ordenado (*índice escaso o no denso*). En el caso de índices totales, el archivo puede estar desordenado, a continuación la definición formal:

DEFINICIÓN: Un **índice total o denso** es aquel que contiene por lo menos una entrada de datos por cada registro del archivo de datos.

DEFINICIÓN: Un **índice escaso o no denso** es aquel que tiene una sola entrada de datos por cada bloque (sector) de disco del archivo de datos.

Los índices totales o densos no suelen utilizarse solos sino en conjunto con otros índices, de esta manera pueden almacenarse en memoria principal obteniendo así una mayor rapidez de acceso.

La siguiente figura muestra la representación de un índice denso y de un índice escaso.

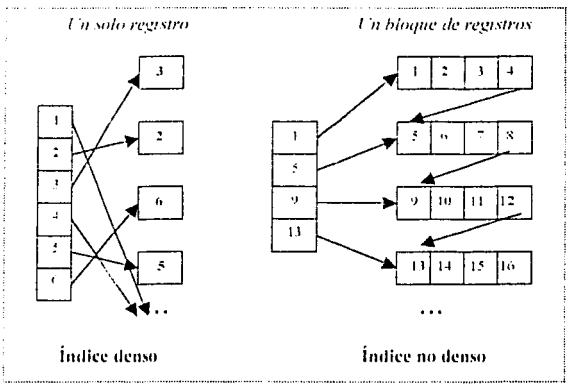


Figura 3.8 Muestra la manera en que los registros se indexan mediante un índice denso y un índice escaso.

A continuación se abordan las tres formas principales en las que se pueden clasificar a los índices:

Índices de un solo nivel

Dentro de los índices que se consideran de un solo nivel los valores de dichos índices deben estar ordenados, ésta es una característica que identifica a los mismos como tales.

Hay varios tipos de índices de un solo nivel, la siguiente división se hace tomando en cuenta el número de claves que tienen los registros.

- *Índices primarios*
- *Índices secundarios*

DEFINICIÓN: Un **índice primario** es un archivo ordenado por una sola clave (llamada clave primaria), cuyos registros son de longitud fija.

Un índice primario es un ejemplo de índice disperso.

DEFINICIÓN: Un **índice secundario** es un archivo ordenado por más de una clave.

Un índice secundario es más grande que un índice primario ya que tiene mayor número de entradas de datos, sin embargo la mejora que se introduce en el tiempo de búsqueda para un registro cualquiera es mayor para el índice secundario que para el índice primario.

Un índice primario provee un único apuntador en contraste con los índices secundarios que proveen muchos apuntadores que pueden hacer referencia a registros, cubos o fragmentos de lista (célula).

Índices multinivel

Un índice multinivel está formado por varios índices ordenados, los cuales cada uno por separado representan un nivel. Cada uno de los índices que forman un índice

multinivel se va construyendo a partir del nivel inmediato anterior, así el segundo nivel del índice se construye a partir del primer nivel y así sucesivamente.

Los índices multinivel reducen el número de acceso a un bloque de disco cuando se busca un registro a partir del valor de su campo de indexación.

La figura 3.9 muestra la manera en la que se relacionan los diferentes niveles de un índice de niveles múltiples.

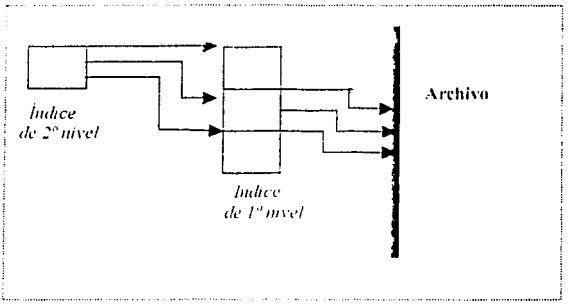


Figura 3.9 Muestra la manera en la cual interactúan los diferentes niveles de un índice que forman un índice multinivel.

Los índices de más alto nivel proporcionan la posición de los índices de menor nivel, a su vez estos proporcionan la posición del bloque donde se encuentran físicamente almacenados los registros.

Índices basados en árboles (trees)

Un árbol pertenece a las estructuras de datos no lineales, y la razón para darlos a conocer como una herramienta para acelerar el acceso a los datos es debido a la gran eficiencia y al hecho de ser utilizado por los actuales sistemas manejadores de bases de datos. La definición formal de un árbol es la siguiente:

DEFINICIÓN: Un **árbol** es una estructura de datos no lineal que está formado por un conjunto finito de nodos, de los cuales se distingue un nodo inicial llamado raíz y el resto de los nodos son conocidos como hijos.

Una de las características más distinguibles es que solo se puede llegar desde la raíz a cualquier nodo hijo a través de un único camino. Cada nodo hijo es a su vez un subárbol.

Todos los nodos excepto la raíz tienen un sucesor. El grado de un árbol indica el número máximo de hijos que podrá tener cualquier nodo. Así un árbol binario es aquel que como máximo puede tener dos hijos en cada nodo.

Estos son conceptos para poder entender como a partir de una estructura de datos se puede construir un índice. La figura 3.10 presenta un árbol.

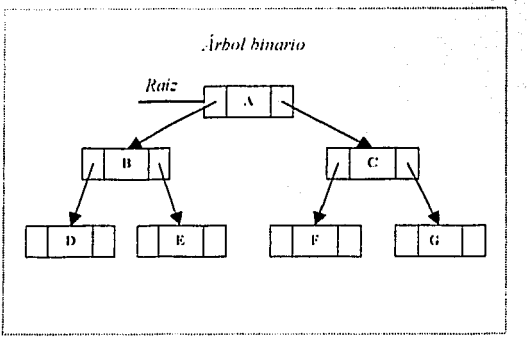


Figura 3.10 Muestra un árbol binario (todos los nodos tienen dos hijos), en el que cada nodo es representado como un registro.

La forma en que esta estructura de datos se utiliza para encontrar un registro es a través de los valores del campo de indexación que contiene cada nodo, así mediante estos valores ubican el siguiente nodo continuando este proceso hasta encontrar finalmente el bloque que contiene el registro buscado.

Los árboles B (binarios) y los árboles B+(balanceados) son árboles de búsqueda, un árbol de búsqueda es un tipo de árbol que sirve para guiar la búsqueda de un registro, previamente teniendo un valor de uno de sus campos.

Es importante mantener equilibrados los árboles de búsqueda porque esto garantiza que no habrá nodos en niveles muy profundos que requieran muchos accesos a bloque durante una búsqueda, además las eliminaciones de registros pueden hacer que algunos nodos queden casi vacíos conllevando a desperdicio de espacio.

Existen dos métodos para recorrer los árboles:

1. **Top down** (*arriba-abajo*). Es método consiste en iniciar el recorrido del árbol a partir de la raíz, continuando con el siguiente nivel y así sucesivamente hasta llegar al último nodo; cuando hay más de un nodo por nivel primero el recorrido se empieza por la izquierda.

2. **Bottom up** (*abajo-arriba*). Este método inicia el recorrido en el último nivel del árbol, cuando hay más de un nodo por nivel el recorrido se inicia por la izquierda el recorrido termina cuando se opera la raíz.

Índices basados en árboles binarios(árboles B)

Ya se ha mencionado la definición de un árbol binario, ahora se definirán propiedades importantes de éstos que son de mucha utilidad para la construcción de un índice.

Propiedades:

- Un índice se forma construyendo un árbol binario con las claves del índice, éste posteriormente se almacena en la memoria secundaria.
- El número de hojas en cada subárbol se encuentra relacionado con la capacidad de una página de memoria.
- Un nodo puede referenciar m nodos hijos como máximo, siendo m el grado del árbol.
- Un nodo hijo contiene como máximo m-1 claves.

Para acceder a los árboles binarios se cuenta con los dos métodos de acceso antes mencionados y además:

1. **Preorden** (raíz-hijo izquierdo-hijo derecho)
2. **Enorden** (hijo izquierdo-raíz-hijo derecho)
3. **Postorden** (hijo izquierdo-hijo derecho-raíz)

Operaciones sobre árboles B:

Inserción

La inserción en un árbol B se fundamenta en el proceso de División y Promoción (staging). La figura 3.11 muestra una inserción en un árbol B.

- Una nueva clave siempre se intenta insertar en una nodo hijo.
- Si éste se encuentra completo, se elige una clave que divida el nuevo conjunto de claves en dos subconjuntos equilibrados.

- Los dos subconjuntos se almacenan en dos páginas, mientras que la clave elegida se inserta en el nivel superior, en donde se puede repetir el proceso.

Supresión (borrado)

La supresión se fundamenta en la Eliminación, Redistribución y Concatenación, eliminándose siempre claves situadas en los nodos hijos. La figura 3.11 muestra esta operación.

- Siempre se debe eliminar claves situadas en un nodo hijo.
- Si se desea eliminar una clave que no esté en un nodo hijo, se intercambia con la clave siguiente, que se encuentra en nodo hijo, y luego se elimina.
- Cuando un nodo hijo queda con un número de claves menor que el mínimo, hay que examinar sus nodos hermanos (que se encuentran al mismo nivel).
- Si un nodo hermano tiene más del número mínimo de claves, se deben redistribuir sus claves con el nodo actual.
- En caso contrario, se concatena el nodo actual, un nodo hermano y la clave que las separa, repitiéndose el proceso.

Debemos suponer que tenemos un bloque con 6 entradas (registros), representados por las letras M, H, B, R, Q, N, V; sobre los cuales se insertarán y se borrarán algunos registros.

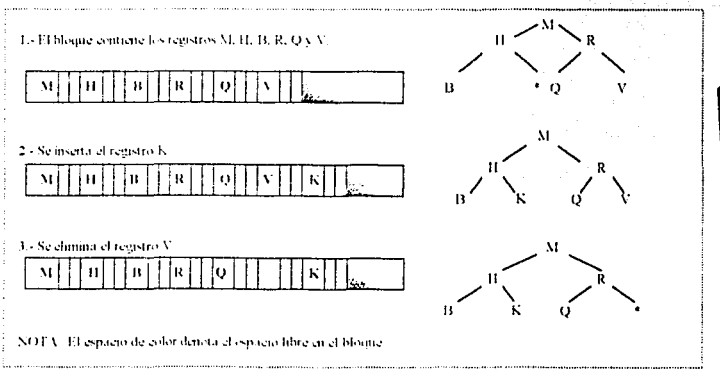


Figura 3.11 Muestra un bloque de entradas de índice organizado a modo de árbol binario después de insertar y eliminar registros.

Índices basados en árboles balanceados (B+ tree)

Pudiera decirse que construir un índice con base a un árbol balanceado es la mejor manera de acelerar el acceso a los datos; una de las razones para asegurar esto es sabiendo que los sistemas manejadores de bases de datos como DB2, Informix, SQL Server, Oracle y Sybase usan índices basados en árboles balanceados para la búsqueda de los datos contenidos en una base de datos.

Los árboles balanceados se desarrollaron con el objetivo de proporcionar un método eficiente para el mantenimiento de índices de varios niveles.
DEFINICIÓN: Un **árbol balanceado (B+ tree)** es un árbol de búsqueda considerado una estructura dinámica de indexación, en la cuál los nodos internos dirigen la búsqueda y los nodos hojas contienen las entradas de los datos.

Propiedades:

- Las operaciones de inserción y de borrado permanecen balanceadas.
- Para la búsqueda de un registro solo se necesita llevar a cabo un recorrido desde la raíz al nodo en cuestión.
- Todos los caminos desde la raíz a las hojas tienen la misma longitud.
- Las claves de los nodos se ordenan por valor $k1 < k2 < k3...$
- Los árboles B+ llevan a cabo el manejo de datos de un modo secuencial y directo.

Operaciones:

Inserción
La inserción utiliza la ***división de bloques***. El procedimiento general consiste en hallar un nodo donde se pueda insertar el valor de la clave, si este nodo se encuentra lleno, se divide en dos nodos, ordenando ascendentemente las claves existentes al igual que la nueva; las claves se ubican de izquierda a derecha, la mitad en cada nuevo nodo que surgió a partir de la división.

Eliminación
La eliminación utiliza la ***redistribución***. El método para eliminar una clave es el proceso inverso que se lleva a cabo en la inserción. En el caso más simple una eliminación no necesita ningún cambio en la estructura del árbol, pero de lo contrario si es requerida una modificación en la estructura del índice. La representación de un árbol balanceado (B+) es la que se presenta en la figura 3.12, donde el número de secciones de cada cuadro representa una clave.

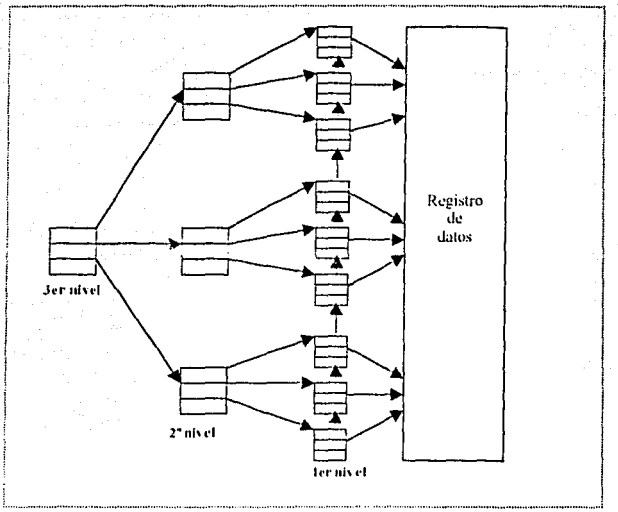


Figura 3.12 Muestra un índice en árbol balanceado.

El índice como un archivo principal debe organizarse de modo que admita inserciones, eliminaciones y que el mantenimiento no requiera mucho tiempo. Un árbol balanceado es la estrategia más frecuentemente usada para el manejo eficaz de las operaciones de inserción y eliminación.

El presente capítulo y el anterior contienen la teoría que retoman y perfeccionan algunas compañías para crear un DBMS comercial dándole un enfoque específico.

En este trabajo se eligió el RDBMS ORACLE para ejemplificar esta cuestión, en el siguiente capítulo se explica la manera en la que este producto tan comercial lleva a cabo el almacenamiento de los datos en una base de datos.

CAPITULO IV

ESTRUCTURAS DE ALMACENAMIENTO EN ORACLE

Este último capítulo hace la comparación entre la teoría que existe respecto al almacenamiento y acceso físico de los datos a una base de datos a través de un sistema manejador de base de datos y lo que en realidad es utilizable en los DBMS actuales, para llevar a cabo esta comparación se eligió el sistema manejador de bases de datos relacional ORACLE. Por tanto se hace un enfoque preciso en dar a conocer las estructuras lógicas (tablespaces, bloques de datos, extents y segments), las estructuras físicas DATA FILES, CONTROL FILES y REDO LOG FILES y la relación existente entre ambas para llevar a cabo el almacenamiento de los datos. Incluyendo también parámetros de almacenamiento, identificadores únicos de cada uno de los renglones (ROWID), el manejo de índices desde el punto de vista del manejador, etc.

Historia de ORACLE

En 1977 Lawrence Ellison, Bob Miner y Ed Oates fundadores de la compañía ORACLE Corporation dan a conocer un sistema manejador de bases de datos relacional (RDBMS) llamado ORACLE. Este RDBMS estaba basado en SYSTEM/R, el primer sistema manejador de bases de datos relacional desarrollado por IBM. [PAGW, 2001]

El impacto de este manejador en el mercado se ve reflejado en el lugar que ocupa actualmente en el mercado mundial, ya que a más de 25 años de su fundación continúa en el liderato de los proveedores de RDBMS. Siendo el principal proveedor de software para el manejo de la información y la segunda empresa de software más grande del mundo.

Para adentrarnos en el almacenamiento de los datos con ORACLE es de suma importancia hacer una distinción entre una "*instancia* ORACLE" y una "*base de datos ORACLE*". Ya que el objetivo principal de este capítulo se centra en las estructuras de almacenamiento en ORACLE, las cuales radican en una base de datos.

Conceptos básicos de una instancia y de una base de datos ORACLE

El término ***base de datos*** se refiere al almacenamiento físico de la información y una ***instancia*** es el software que se ejecuta en una máquina (servidor) que provee el acceso a la información contenida en la base de datos.

Una base de datos ORACLE es **física**, es decir consiste en una serie de archivos (*files*) almacenados en dispositivos de memoria secundaria (disco duro). Una instancia ORACLE es **lógica**, es decir, consiste en un conjunto de estructuras de memoria y procesos. Si bien, la explicación de cada una se hace por separado más adelante se muestra la manera en la que se relacionan en el almacenamiento de los datos.

Cuando un usuario se conecta a ORACLE, en realidad se conecta a la instancia creando un proceso de usuario, el cual le permite accesar a la base de datos. Una instancia únicamente conecta a una base de datos. En las siguientes líneas se distinguen los componentes tanto de una instancia como de una base de datos ORACLE.

DEFINICIÓN: Una **instancia ORACLE** matemáticamente se expresa como,
Instancia ORACLE = SGA (*System Global Area*) + Procesos de background.

Cada una de estas dos áreas que forman una instancia ORACLE se compone a su vez de otras áreas.

I. Área global del sistema (SGA, System Global Area)

- Conjunto compartido (*Shared pool*)
- Caché de buffers de base de datos (*database buffer cache*)
- Buffer de redo log (*redo log buffer*)
- Conjunto de Java (*Java pool*) *
- Área global de proceso (PGA, *Process Global Area*)

II. Procesos de background

1. SMON (System Monitor)
2. PMON (Process Monitor
3. DBWR (Database Writer)
4. LGWR (Log Writer)
5. CKPT (Checkpoint Process)
6. RECO (Recover)
7. ARCH (Archiver)

* De la versión 8i en adelante.

DEFINICIÓN: Una **base de datos ORACLE** matemáticamente se expresa como:
BD ORACLE = Estructuras físicas + Estructuras lógicas

La fórmula está compuesta de dos áreas: la primera en donde los objetos se utilizan de forma interna por parte del RDBMS, que podemos denominar *"Objetos de la base de datos utilizados por el sistema"*, y hacen referencia a las **estructuras físicas de la base de datos**; y la segunda, en la cual los objetos pueden accesar a cualquier proceso de background de una instancia, denominados *"Objetos de la base de datos utilizados por el usuario"* o **estructuras lógicas de la base de datos**. Cada una de estas áreas integra a su vez diversas estructuras, la figura 4.1 muestra cada uno de los componentes que forman una instancia y una base de datos ORACLE.

I. Estructuras físicas:

1. Data files
2. Control files
3. Redo log files

II. Estructuras lógicas:

1. Tablespaces
2. Segments
3. Extents
4. Bloques (data blocks)

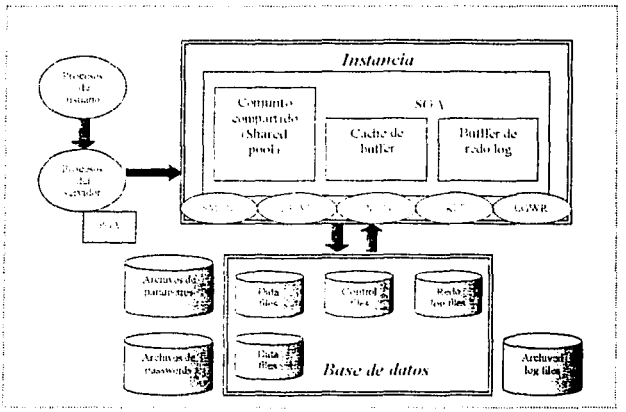


Figura 4.1 Muestra la manera en que se relacionan una instancia y una base de datos ORACLE.

En la figura se ubican tres archivos que no son parte de la base de datos:

- Archivos de parámetros (*Parameter files*): Definen las características de una instancia ORACLE.
- Archivos de password (*Password files*): Verifica la autenticación de los usuarios.
- Archivos de archived log (*Archived log files*): Son copias de los archivos de redo log, que pueden ser útiles para la recuperación a fallas.

Estructuras lógicas de almacenamiento de una base de datos ORACLE.

Anteriormente se identificaron las estructuras lógicas en las cuales se almacenan los datos, ahora es momento de profundizar en cada una de ellas.

Una manera de entender lo que son tales estructuras es concibiéndolas como objetos que los usuarios accesan y que los desarrolladores crean. Así por ejemplo cuando se requiere información sobre la tabla EMPLEADOS, los usuarios accesan a dicha tabla que previamente fue creada por un desarrollador.

Tablespace

Una base de datos de ORACLE está lógicamente formada por áreas de espacio conocidas como *tablespace*.

Un *tablespace* está formado de archivos del sistema operativo los cuáles son considerados estructuras físicas de almacenamiento y son llamados *datafiles* (se abordan con detenimiento en el siguiente tema), así un *tablespace* puede consistir en uno o más *datafiles*, contrariamente sucede con los *datafiles* pues solo pueden pertenecer a solo un *TABLESPACE*.

Un *tablespace* contiene los diferentes tipos de objetos que se almacenan en una base de datos: tablas, vistas, funciones, paquetes, etc., debido a esto cuando se crean objetos se debe especificar el *tablespace* en el que se crearán.

Una de las razones para definirlo como la unidad fundamental de almacenamiento corresponde al papel que desempeña pues es el vínculo con las estructuras físicas de almacenamiento (*datafiles*). La siguiente figura muestra la relación entre tablespaces y *datafiles*.

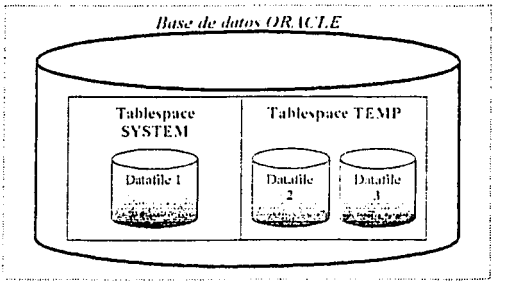


Figura 4.2 Muestra la relacion entre los *tablespaces* y los *datafiles.*

ORACLE crea tres tablespaces por default, estos son:

- SYSTEM: En este tablespace se almacena el diccionario de datos.
- TEMP: En este se almacenan objetos temporales (tablas, índices, etc.).
- ROLLBACK: En este se almacenan las transacciones de deshacer.

Hay otras estructuras lógicas como los bloques de datos, extents y segments que en conjunto con los tablaspaces sirven como unidades de almacenamiento de los datos.

Bloques de datos, extents y segments.

ORACLE asigna espacio para todos los datos en una base de datos. Las unidades lógicas de almacenamiento son: **bloques de datos, extents y segments.** La figura 4.3 define la relación entre estas unidades lógicas de almacenamiento.

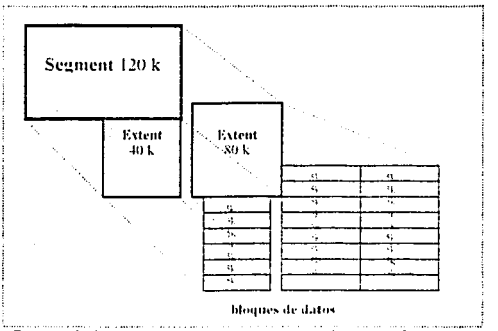


Figura 4.3 Muestra como los bloques de datos forman un extent y como varios de estos a su vez forman un segment.

Un **bloque de datos** también es llamado *bloque lógico, bloque ORACLE o página*, es el nivel inferior de almacenamiento en una base de datos que se puede manipular. Cualquier acceso a datos se realiza en términos del *bloque de datos*. El tamaño del *bloque de datos* es el número de bytes que lee y escribe el RDBMS en los archivos de datos.

Es importante mencionar que no es lo mismo un *bloque de datos* ORACLE que un bloque del sistema operativo; aunque un *bloque de datos* está compuesto de bloques del sistema operativo, no son lo mismo. Es recomendable que el tamaño de un bloque de datos sea un múltiplo del bloque del sistema operativo, así por ejemplo en unix comúnmente es 8k. Cada bloque de datos consta de tres áreas:

- **Cabecera**: La cabecera de un bloque de datos contiene información general acerca del mismo, como tipos de datos del segment que se almacena en el bloque, la dirección del bloque, directorio de tabla (*table directory*) el cual indica si una tabla tiene renglones en este bloque, directorio renglón (*row directory*), el cual proporciona información sobre los renglones almacenados en dicho bloque. Una cabecera de bloque suele utilizar de 85 a 100 bytes en el bloque (Fig. 4.4).
- **Espacio libre**: Cada bloque de datos cuenta con un área de espacio libre, la cual se ubica a la mitad del bloque ya que debido a esta posición permite el crecimiento tanto de lo cabecera como del área donde se encuentran los datos. Esta área es usada para insertar renglones nuevos o actualizar los ya existentes. El manejo de este espacio esta a cargo de los parámetros PCTUSED y PCTFREE (Fig.4.4).
- **Datos**: En esta área se almacenan los datos (en renglones)que contiene el bloque (Fig. 4.4).

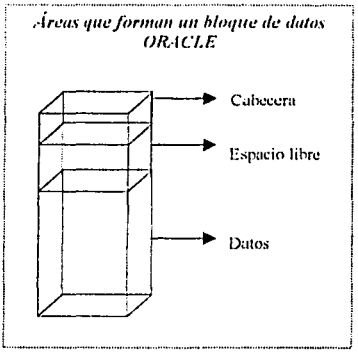


Figura 4.4 Muestra las áreas de contenido de un bloque de datos

Pctfree y pctused

En la explicación del área de espacio libre se hace referencia a dos parámetros que administran dicho espacio, estos son ***pctfree y pctused.*** Estos parámetros permiten controlar el uso del espacio libre ya sea para inserciones o para actualizaciones.

El proceso de guardar información en la base de datos consiste primeramente en encontrar uno o más bloques en los extents asignados de un segment, para facilitar esta tarea ORACLE guarda un listado de bloques libres para cada segment y es aquí donde son imprescindibles los parámetros *pctfree* y *pctused* para determinar si los bloques tienen o no espacio suficiente para almacenar nueva información.

El parámetro *pctfree* representa el porcentaje de espacio de bloque que se reserva para actualizaciones de los datos. Por ejemplo, si un bloque tiene un valor *pctfree* de 20%, el 80% del espacio restante se utiliza para almacenar nuevos renglones. Una vez que se satura este 80% el bloque en cuestión se retira de la lista de bloques disponibles para almacenar datos. La lista de bloques libres utiliza el 20% restante del espacio para controlar las actualizaciones de renglones dentro del bloque. El porcentaje por default es 10%.

El parámetro *pctused* especifica la cantidad de espacio que se debe liberar dentro de un bloque antes de ser reintegrado a la lista de bloques disponibles para poder seguir almacenando datos en dicho bloque. Continuando con el ejemplo anterior, tenemos *pctfree* de 20% y *pctused* de 40%. Podemos utilizar entonces el 80% del porcentaje restante del espacio libre, una vez que saturamos ese 80% mediante inserciones o actualizaciones que alteran el tamaño de un renglón, este bloque pasa a la lista de bloques no disponibles hasta que disminuya el porcentaje de espacio utilizado a menos del 40% indicado por el parámetro *pctused*, esto va a ser posible mediante la eliminación de renglones o la actualización de los mismos en la que los nuevos valores ocupen menos espacio. Cuando se llega a tener nuevamente espacio para almacenar los datos el bloque se reintegra a la lista de bloques disponibles o libres. El porcentaje por default es de 40%.

Los parámetros *pctfree* y *pctused* funcionan conjuntamente para asegurar disponibilidad de espacio en un bloque. Los valores de dichos parámetros no deberían sumar el 100%, si llegara a ocurrir esta situación es posible que dicho bloque se elimine y se ubique continuamente en la lista de bloques disponibles a la mínima manipulación.

Las siguientes líneas explican como se hace el almacenamiento de los renglones que pertenecen a las tablas.

Almacenamiento de renglones en un bloque de datos.

Un renglón es almacenado en una base de datos como un archivo de longitud variable. Las columnas de un renglón son generalmente almacenadas en el orden en el cuál fueron definidas durante la creación de la tabla. Un renglón está dividido en dos secciones que lo identifican como único:

- ***Cabecera,*** la cuál contiene información del renglón como el número de columnas y el estado en el que se encuentra el renglón (si esta encadenado o migrado). Fig. 4.5.
- ***Datos,*** en esta área se almacenan la longitud y el valor de las columnas que integran un renglón. Solo un byte se necesita para almacenar la longitud de la columna si esta no excede de 250 bytes, el valor de la columna se almacena continuamente a la longitud de la misma. Fig. 4.5.

Cuando se insertan renglones de tamaño muy grande en una tabla (renglones que contienen textos muy grandes) no es factible almacenar dicho renglón en el mismo bloque de datos. Este conflicto se soluciona almacenando el renglón en una cadena (chain) de bloque de datos, la cual consiste en una vinculación dos o más bloques. Este proceso es conocido como **Encadenamiento de renglón** (*Row Chaining*).

Existe una complicación también en el proceso de actualización de un renglón que llega a ser muy grande ya que debido a este gran tamaño es insuficiente el espacio en un bloque para poder llevar a cabo la actualización del renglón. Esta cuestión se soluciona mediante una **Migración de renglón** (*Row Migration*), la cual ORACLE lleva a cabo cuando el renglón en cuestión es movido a un nuevo bloque y se deja un apuntador del bloque original a la nueva ubicación del renglón.

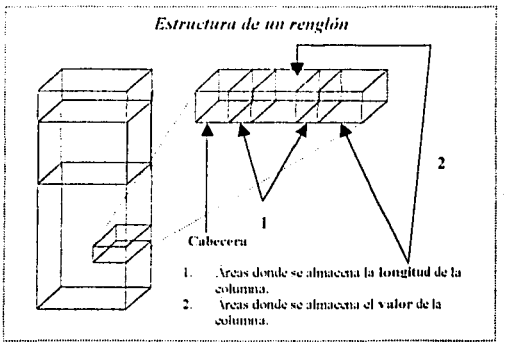


Figura 4.5 Muestra la manera en la que un renglón se almacena en un bloque de datos.

Un bloque es la unidad más pequeña en la que se almacenan los datos, un conjunto de bloques forman un extent.

Un **extent** es una unidad lógica de almacenamiento compuesta por uno o más bloques de datos contiguos, a su vez uno o más extents forman un **segment.**

Los extents que pertenecen a un segment determinado pueden ser del mismo tamaño o de tamaños diferentes. A un segment se le asignan extents en el momento de crear los objetos en la especificación del comando *CREATE.*

En realidad los segments son los objetos almacenados en la base de datos que son creados por el usuario final y por el usuario que administra la base de datos (DBA, *Data Base Administrator*).

Una base de datos puede contener cuatro diferentes tipos de segments:

I. **Segmentos de datos (tabla, cluster e índice)**
II. **Segmentos de rollback**
III. **Segmentos temporales**

Segmentos de datos

Tablas

Dentro de los segmentos de datos se encuentran las **tablas,** las cuales son las unidades lógicas más comunes de almacenamiento ya que en ellas se almacenan los datos. Cada tabla está compuesta de una o más columnas, cada una de las cuales tiene asignado un nombre y un tipo de datos. Los datos que constituyen una tabla al momento de almacenarse no siguen un orden. Todos los datos pertenecientes a una tabla deben ser almacenados solamente en un tablespace. ORACLE utiliza la función de HASH para almacenar datos en una tabla, se aborda al concluir la explicación de las estructuras físicas de almacenamiento.

Clusters

Un **cluster** (agrupamiento de tablas) también es un segmento de datos. Los clusters son grupos de una o más tablas almacenadas físicamente juntas debido a que tienen en común uno o varias columnas. Los renglones en un cluster son almacenados basándose en valores claves de columnas. Las tablas en un cluster pertenecen al mismo segmento y comparten las mismas características de almacenamiento.

Índices

Los **índices,** al igual que las tablas y los clusters son un segmento de datos, su función es agilizar y acelerar el acceso a datos específicos que se encuentran en las tablas. Un índice contiene el valor para una o más columnas de una tabla y el ROWID para los valores de columna correspondientes. Cuando ORACLE necesita buscar un renglón específico dentro de una tabla busca en el índice el ROWID y después obtiene los datos directamente de la tabla.

DEFINICIÓN: Un **ROWID** (*Row IDentifier, Identicador de renglón*) es una estructura interna (seudo-columna) propia de ORACLE que identifica de forma única a cada uno de los renglones que contiene una tabla. Es un valor que permite localizar un renglón.

La manera más rápida de acceder a un renglón es a través del ROWID. Un ROWID necesita diez bytes (80 bits) de almacenamiento en disco y se despliega usando 18 caracteres. Consta de los siguientes componentes (figura 4.6):

1. **Número de objeto de datos.** Este número se le asigna a cada objeto (tabla o índice) cuando es creado, es un número único dentro de una base de datos. Necesita 32 bits.
2. **Número relativo de archivo.** Es un número único que se le asigna a cada archivo dentro de un tablespace. Necesita 10 bits.
3. **Número de bloque.** Este número representa la posición del bloque que contiene el renglón dentro del archivo. Requiere 22 bits.
4. **Número de renglón.** Este número identifica la posición de un renglón en el bloque donde reside. Necesita 16 bits.

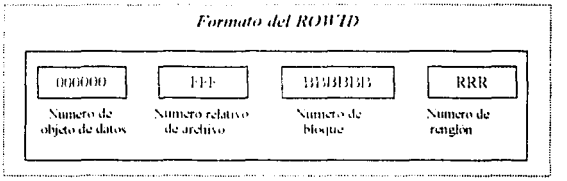


Figura 4.6 Muestra los componentes que forman un ROWID.

La notación del formato está expresada en código base-64. Se usan por lo tanto, seis posiciones para el número de objeto de datos, tres para el número relativo de archivo, seis para el número de bloque y tres para el número de renglón. El código base-64 usa todos los caracteres del alfabeto en mayúsculas y minúsculas, (A-Z, a-z) los números del 0 al 9 y los signos "+ y /". Ejemplo: AAADC4AACAAAAMAAAG.

- AAADC4 es el número de objeto de datos
- AAC es el número relativo de archivo
- AAAAMA es el número de bloque
- AAG es el número de renglón

El ROWID desempeña un papel muy importante en el proceso de aceleración de búsqueda de los datos, es decir dentro de los índices. ORACLE maneja diversos tipos de índices, los clasifica en lógicos y físicos.

I. Índices lógicos

a. Una columna o concatenados
b. Únicos o no únicos
c. Basados en función

II. Índices físicos

a. Particionados o no-particionados
b. Árbol balanceado(B-tree)
c. Normal o de clave inversa
d. Bitmap

La clasificación lógica de los índices agrupa índices desde una perspectiva de aplicación mientras que la clasificación física es derivada de la manera en que los índices son almacenados, estos últimos son objeto de estudio en el presente trabajo.

Índices físicos

Los **índices particionados** son usados por tablas muy grandes para llevar a cabo el almacenamiento de las entradas de los índices en varios segments. El particionamiento permite a un índice ser esparcido en varios tablaspaces. Este tipo de índices es frecuentemente usado con tablas particionadas.

Otro tipo de índice físico son los **árboles balanceados (B-tree),** los cuáles se caracterizan por almacenar una lista de ROWIDS para cada clave. La estructura de este índice es de acuerdo a la estructura de un árbol balanceado ya que igualmente tiene una raíz en el nivel más alto la cual apunta al siguiente nivel en el que se encuentran los bloques ramas y a su vez éstos bloques ramas apuntan al nivel más bajo de la estructura en donde se encuentran los nodos hojas los cuales a diferencia de los dos niveles anteriores se encuentran doblemente ligados para facilitar el barrido del índice. La siguiente figura muestra la estructura en cuestión.

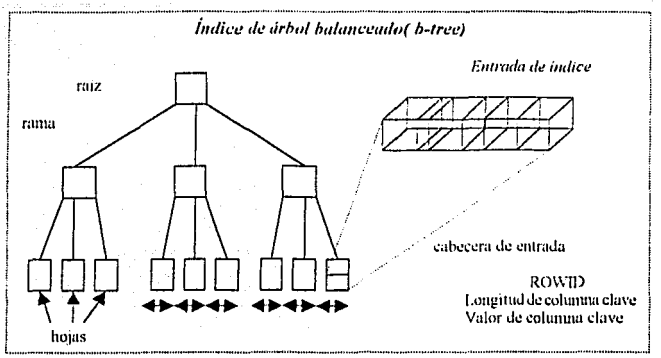


Figura 4.7 Muestra la estructura de un índice de árbol balanceado(B-tree).

En esta figura se distinguen varios componentes de una entrada de índice:

- *Cabecera de entrada*, esta sección almacena el número de columnas que integran el índice.
- *Longitud y valor de la columna clave*, en estas áreas se encuentran el tamaño de una columna en la clave seguida del valor para la columna.
- *ROWID* de un renglón, este contiene los valores de la clave.

El siguiente tipo de índice es el **índice de clave inversa,** llamado así debido a que invierte los bytes de cada columna indexada a excepción del ROWID. Por ejemplo si se inserta la matricula 7698 de un empleado, el valor de la clave inversa que se almacena en el índice es 8967 (figura 4.8).

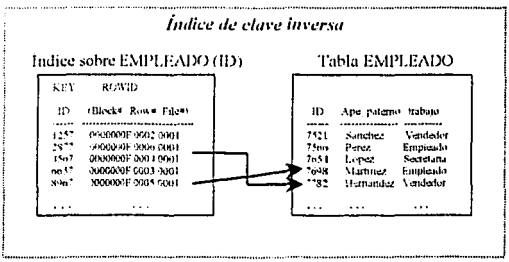


Figura 4.8 Muestra un índice de clave inversa sobre la tabla EMPLEADO.

El cuarto tipo de índice físico es el **índice de mapa de bits** (*Bitmap*), este índice recientemente ha sido usado. Un índice de mapa de bits consiste en crear un mapa de bits que debe incluir los valores de columna dentro de la tabla indexada, dichos valores se almacenan en el índice. En otras palabras, el índice contiene un mapa de bits para cada renglon dentro de la clave, cada mapa contiene un bit por cada renglón dentro de la tabla. El bit es 1 si el valor está contenido dentro de la fila y 0 si no está.

Los índices de mapa de bits pueden ser más eficientes que los índices de árbol balanceado cuando:

1. Una tabla tiene muchos renglones (millones).
2. Las columnas de una tabla tienen baja cardinalidad (número pequeño de valores distintos).
3. En una consulta se usan muchas cláusulas WHERE que involucran el operador OR.

La estructura de un índice de mapa de bits es muy similar a la de un índice de árbol balanceado, solo que en el primero los nodos hojas almacenan un mapa de bits por cada valor clave en lugar de una lista de ROWID. Se puede decir que cada bit en el mapa de bits le corresponde un ROWID en el árbol balanceado (Fig. 4.9).

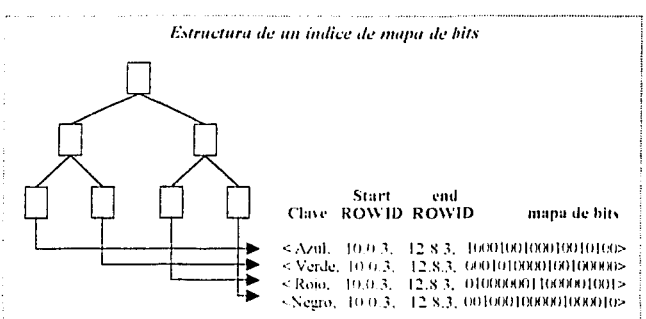


Figura 4.9 Muestra los componentes de un índice de mapa de bits

La estructura de un índice de mapa de bits se presenta sobre una tabla llamada COCHE, en la cual las claves son los colores de los coches. Cada nodo hoja contiene los siguientes componentes:

- *Clave*, esta contiene la longitud y el valor cada clave columna, en el ejemplo de la figura 4.10 la clave consta de una sola columna y la primera entrada tiene la clave azul.

- *Start ROWID*, es el ROWID del primer renglón, en el ejemplo el número 10 hace referencia al número de bloque, el 0 al número de renglón y el 3 al número de archivo.
- *End* ROWID, es un apuntador al último renglón en la tabla. En el ejemplo el 12 indica el número de bloque, el 8 el número de renglón y el 3 el número de archivo.
- *El mapa de bits*, como anteriormente se explicó hay un mapa de bits para cada renglón, en el ejemplo la tabla COCHE contiene 19 renglones, los cuales son representados por un bit (1 o 0) de acuerdo al color de los renglones. Así por ejemplo los renglones 1,5,8,12,15 y 17 son coches azules, los renglones 4,6,11 y 14 son verdes, los renglones 2,9,10,16 y 19 son rojos y el resto son negros.

La estructura del índice de mapa de bits crea índices más pequeños que el índice de árbol balanceado, pero como se indicó se reservan para ciertos tipos de datos.

Cuando se crea algún segmento de datos, se debe hacer una serie de especificaciones acerca de su almacenamiento. Tales especificaciones son conocidas como "*parámetros de storage*", dichos parámetros le indican a ORACLE como efectuar el almacenamiento de tablas, índices, etc.

Parámetros de Storage

Mediante los parámetros de storage es posible controlar el acceso a datos y el espacio en la base de datos.

En el momento de crear algún segmento de datos es conveniente especificar los valores de cada uno de los parámetros de storage que debe tener cada uno de los objetos mencionados, si algún parámetro no se especifica, ORACLE asume para el nuevo objeto que se crea el valor del parámetro especificado para el tablaspace en el cuál fue creado el objeto en cuestión.

Los parámetros de storage pueden ser usados en cualquiera de las siguiente sentencias:

- CREATE CLUSTER y ALTER CLUSTER
- CREATE INDEX y ALTER INDEX
- CREATE ROLLBACK SEGMENT y ALTER ROLLBACK SEGMENT
- CREATE TABLE y ALTER TABLE
- CREATE TABLESPACE y ALTER TABLESPACE

Como parámetros de storage se encuentran:

1. INITIAL
2. NEXT
3. MINEXTENTS
4. MAXEXTENTS
5. PCTINCREASE

Cada uno de estos parámetros se describe a continuación.

1. INITIAL

El parámetro INITIAL especifica en bytes el tamaño del primer extent del objeto que se pretende almacenar. El valor por default es el tamaño de cinco bloques de datos ORACLE. Este parámetro no se puede usar en el comando ALTER.

Se puede especificar el tamaño de este parámetro y del parámetro NEXT en Kilobytes y en Megabytes.

2. NEXT

Este parámetro especifica el tamaño (en bytes) del siguiente extent en el cuál se almacena un objeto. El valor por default es el tamaño de cinco bloques de datos ORACLE, mientras que el valor mínimo es el tamaño de un bloque de datos.

3. PCTINCREASE

El número que se asigna al parámetro PCTINCREASE especifica el porcentaje con el que crecerán el tercer y subsecuentes extents. El valor por default es 50 (%), esto implica que cada extent a partir del tercero es 50% más grande que el anterior. El valor mínimo es 0, lo que significa que todos los extent después del primero son del mismo tamaño.

Los valores máximos que se les pueden asignar a los parámetros initial, next y pctincrease dependen del sistema operativo.

4. MINEXTENTS

El valor del parámetro MINEXTENT especifica el número total de extent que se asociarán cuando se lleve a cabo la creación de algún objeto. El valor por default es uno, esto implica que ORACLE solo asocia un extent (el extent inicial) al objeto que se construye. Si el valor de este parámetro es mayor a uno, ORACLE calcula el tamaño de los extent subsecuentes con la ayuda de los valores de los tres parámetros anteriores.

5. MAXEXTENTS

El valor del parámetro MAXEXTENTS especifica el número total de extents (incluyendo el primero) que ORACLE puede asociar por el objeto que se pretende crear. El valor mínimo es uno. El valor por default depende directamente del tamaño del bloque de datos ORACLE.

Los parámetros de storage se introducen en una sentencia mediante la palabra STORAGE, en el siguiente ejemplo se crea una tabla especificando parámetros de storage:

```
CREATE TABLE empleados
      (no_emp          NUMBER(3),*
       nombre_emp      VARCHAR2(15),*
       ape_pat_emp     VARCHAR2(15),
       ape_mat_emp     VARCHAR2(15),
       direccion_emp VARCHAR2(25))
       STORAGE (INITIAL 100K  NEXT 50K
                MINEXTENTS 1  MAXEXTENTS 50  PCTINCREASE 5);
```

Dentro del comando STORAGE se define el espacio que ORACLE debe reservar para la creación de la tabla EMPLEADOS de la siguiente manera:

- El valor de parámetro **INITIAL** es 100K, lo que significa que el tamaño del primer extent es de 100 kilobytes.
- Si se almacenan en la tabla una gran cantidad de datos de tal manera que excedan los 100K, ORACLE reservará un segundo extent de 50K de tamaño, pues así se especificó en el parámetro **NEXT**.
- El valor del parámetro **MINEXTENTS** es 1, esto implica que ORACLE reserva un extent para la creación de la tabla.
- El valor del parámetro **MAXEXTENTS** es 50, esto implica que ORACLE reserva como máximo 50 extenst en caso que la tabla crezca.
- Suponiendo que en la tabla se han almacenado muchos datos y se han saturado los dos primeros extents, ORACLE reserva un tercer extent para así continuar almacenando datos en la tabla EMPLEADOS. El tamaño de este tercer extent está determinado en el parámetro **PCTINCREASE**, el cual tiene un valor de 5(%), lo que significa que el tamaño del tercer extent será 5% más grande que el segundo extent, explícitamente tendrá 52.5 Kilobytes más. ORACLE trunca el valor de los bytes dependiendo del tamaño de los bloques de datos.

*Tipos de datos manejados por ORACLE.

Segmentos de rollback

El segundo tipo de segments, son los segmentos de rollback. Un segmento de rollback es una porción de la base de datos que almacena la imagen anterior de los datos (dirección y valor de los datos) involucrados en una transacción. Los segmentos de rollback son utilizados para:

1. **Proporcionar consistencia de lectura,** esto es: cuando se realiza un cambio a los datos, la imagen anterior se copia en los segmentos de rollback realizándose la actualización en los bloques de datos, si algún usuario solicita los datos anteriores, la imagen almacenada en los segmentos de rollback regresa para ser utilizada por el usuario.
2. **Deshacer transacciones,** las cuales se hayan hecho con sentencias SQL y que pertenezcan a los comandos DML (delete, insert, update).
3. **Recuperar la base de datos.** Los segmentos de rollback son usados para deshacer los efectos de transacciones que no han sido guardadas.

Los segmentos de rollback como los segmentos de datos se encuentran en un tablaspace y cada uno está compuesto de al menos dos extents sobre los cuales se escriben las transacciones, si éstas son muy grandes y no caben en un extent asignado previamente se asigna otro extent. Cada base de datos cuenta con un segmento de rollback llamado SYSTEM el cual se crea cuando simultáneamente con la creación de la base de datos. La figura 4.10 muestra la estructura de un segmento de rollback.

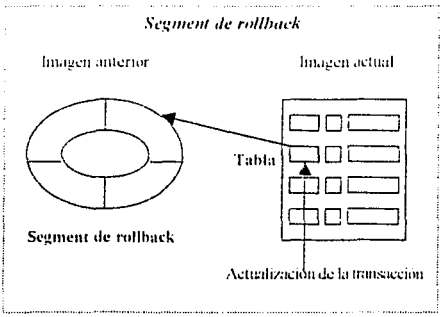


Figura 4.10 Muestra los componentes de un segment de rollback.

Los segmentos de rollback no pueden ser accesados o leídos por los usuarios de la base de datos ni el DBA, son exclusivos del manejador. Una explicación detallada de los segmentos de rollback se encuentra fuera del objetivo de este trabajo, únicamente se proporcionan aspectos muy generales acerca de ellos.

El tercer tipo de segmentos, son los segmentos temporales.

Segmentos temporales

En el procesamiento de las consultas hay ocasiones en las que es necesario un espacio temporal debido a que el espacio en memoria es insuficiente para completar la transacción, cuando se presenta esta situación los resultados de la consulta son escritos en dicho espacio temporal, éste espacio es conocido como **segmento temporal.** No todas las consultas de un segmento temporal se pueden almacenar en estos segmentos, solo los que contengan la siguiente lista de comandos:

- CREATE INDEX
- SELECT ... ORDER BY
- SELECT DISTINCT
- SELECT ...GROUP BY
- SELECT ...UNION
- SELECT ...INTERSECT
- SELECT ...MINUS

La siguiente figura resume la relación que existe entre las estructuras lógicas y la manera en que se vinculan con las estructuras físicas (data files).

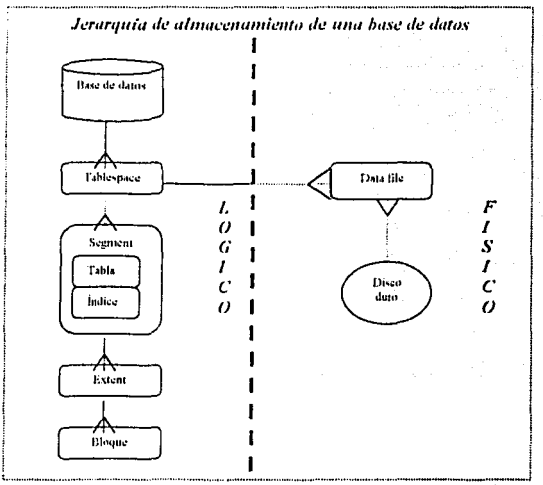


Fig. 4.11 Muestra la vinculación entre lo lógico y lo físico de una base de datos.

Las estructuras lógicas de almacenamiento se encuentran organizadas de acuerdo a una jerarquía específica de almacenamiento en la cual cada estructura (tablespace, segments, extents y bloques) ocupa un lugar en el acceso a los datos.

Es importante poner especial atención en las líneas mediante las cuales se unen las diferentes estructuras de almacenamiento ya que la figura en sí es un diagrama Entidad-Relación en el que las relaciones existentes entre cada una de ellas se encuentran representadas con notación Case* Method. En realidad la línea con dos patas al final representa el grado (uno o más) de la relación entre estructura y estructura. La figura 4.11 lo que expresa es:

- Cada base de datos ORACLE debe estar construida de uno o más tablespaces.
- Cada tablespace debe ser parte de uno y sólo una base de datos ORACLE.
- Cada tablaspace debe ser construido de uno o más datafiles.
- Cada datafile puede ser parte de uno y sólo un tablespace.
- Cada tablaspace puede estar dividido en uno o más segments.
- Cada segment debe estar incluido en uno y sólo un tablaspace.
- Cada segment debe estar formado por uno o más extents.
- Cada extent debe ser incluido en uno y sólo un segment.
- Cada extent debe estar compuesto de uno o más bloques de datos ORACLE.
- Cada bloque de datos ORACLE debe ser parte de uno y sólo un extent.
- Cada datafile debe ser residente en uno y sólo un disco duro.*
- Cada disco duro puede almacenar uno o más data files.

Ya hemos definido todas las estructuras lógicas de almacenamiento, el siguiente tema aborda las estructuras físicas de almacenamiento.

Estructuras físicas de almacenamiento de una base de datos ORACLE

Anteriormente se mencionaron los tres tipos fundamentales de archivos físicos (DATA FILES, CONTROL FILES Y REDO LOG FILES) que envuelven a una base de datos ORACLE. Hay otros archivos que son usados dentro de un ambiente de base de datos, tales como los *archivos de parámetros* y los *archivos de inicialización de una instancia.*

* Un datafile puede estar distribuido en varios discos, dependiendo del sistema operativo sobre el que esté instalado el RDBMS.

El primer tipo de archivos físicos que se aborda es los **DATA FILES** (*Archivos de datos*), como la traducción lo indica son archivos en los cuales se encuentran almacenados todos los datos (diccionario de datos, segments de rollback, tablas, índices, etc.), de manera lógica se almacenan en tablaspaces y de manera física en DATA FILES. Cada tablespace consta de uno o más DATA FILES.

Los DATA FILES trabajan en conjunto con el sistema operativo para llevar a cabo el almacenamiento de los datos dentro de un tablaspace. Un DATA FILE está compuesto de bloques de datos ORACLE, estos a su vez están compuestos de bloques de sistema operativo (figura 4.12).

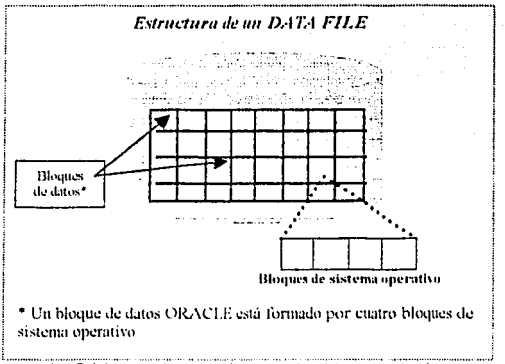


Figura 4.12 Muestra la relación entre los bloques de datos y los bloques del sistema operativo.

Los DATA FILES se crean en SQL mediante alterar un tablespaces se agregan, es decir, no existe un comando CREATE DATAFILE, más bien una vez construido el tablaspace se altera con la siguiente sentencia de comandos ALTER nombre del tablespace ADD ubicación en disco y nombre del DATA FILE con extensión ".ora".

Cuando se crean tablas, índices o cualquier otro segment ocurren dos cosas:

1. Se crean dentro de un tablaspace y
2. Físicamente son almacenados en los DATA FILES asociados al mismo tablaspace donde se han crean los objetos.

Por tanto los DATA FILES son el repositorio para los datos dentro de un tablaspace (figura 4.13).

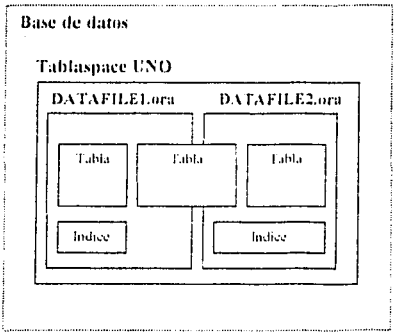


Figura 4.13 Muestra la relación entre un tablaspace y los DATA FILES en el almacenamiento de objetos.

El segundo tipo de archivos físicos es los CONTROL FILES.

Un **CONTROL FILE** de una base de datos es un archivo binario de tamaño pequeño que es necesario para levantar y operar, éstos archivos contienen información acerca del estado de la base de datos como:

- Nombre de la base de datos
- Fecha de creación de la base de datos
- Estado actual de los DATA FILES
- Detalles referentes cuando se levanta y se cierra la base de datos
- Nombres y ubicación de los DATA FILES y REDO LOG FILES de la base de datos
- Nombres de los tablaspaces
- Información de respaldos de la base de datos

Los CONTROL FILES son actualizados continuamente por ORACLE durante el uso de la base de datos, por tanto la información contenida en ellos no puede ser modificada por ninguna persona, únicamente por el manejador.

El tamaño de los CONTROL FILES está sujeto a los parámetros de inicialización: maxlogifles, maxlogmembers, maxloghistory, maxlogdatafiles y maxinstances.

Una base de datos no se puede abrir sin los CONTROL FILES y por consiguiente no se puede tener acceso a los datos contenidos en la base de datos, debido a la importancia de los CONTROL FILES en el funcionamiento de la base de datos es recomendable hacer varias copias de los mismos.

El tercer tipo de archivos físicos es los **REDO LOG FILES**, éstos archivos registran los cambios hechos a una base de datos, son utilizados por ORACLE durante la recuperación de la base de datos.

Los REDO LOG FILES se componen como mínimo de dos *grupos* de REDO LOG FILES, dentro de cada grupo los REDO LOG FILES son llamados *miembros*. Cada grupo puede constar de varios miembros, cada uno de estos es una copia exacta de los demás (figura 4.14). Una base de datos debe tener a lo menos dos grupos de CONTROL FILES. Todos los miembros que integran un grupo tienen el mismo tamaño.

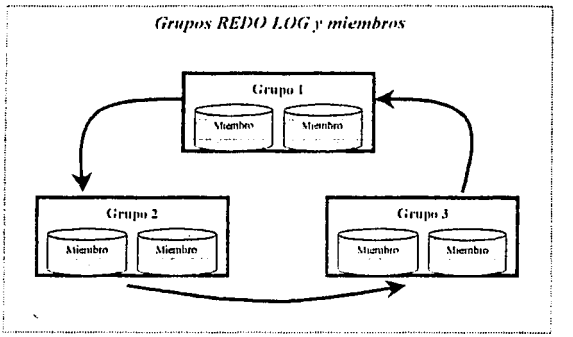


Figura 4.14 Muestra los componentes de los REDO LOG FILES (grupos y miembros)

Los grupos de REDO LOG FILES están en contacto directo con el proceso de brackgraund LGWR (Log writer) ya que es éste el que escribe la misma información a todos los grupos de REDO LOG FILES.

Al abordar el tema de los tipos de segmentos, se hizo mención a tres diferente tipos de segmentos: de datos, de rollback y temporales. Dentro de los primeros se hizo distinción de los objetos que son considerados como segmentos tales son: tablas, cluster, etc. Cuando se definió una tabla se distinguió una manera diferente de almacenar datos en una tabla, esto es mediante la función de HASH.

Almacenamiento de datos con la función HASH

Un camino opcional para almacenar datos en una tabla es usar la **función HASH,** esta función se usa en los clusters HASH y es en realidad mediante estos que se lleva a cabo el almacenamiento de los datos.

Los renglones de una tabla en un cluster HASH son almacenados físicamente tomando en cuenta los resultados de la función HASH. Esta función es usada para generar una distribución de valores numéricos llamados **"valores HASH"** los cuales son basados en un valor clave del cluster pudiendo ser ésta una columna o varias columnas.

Para encontrar o almacenar renglones en un cluster HASH, ORACLE aplica la función HASH a los valores clave del cluster y como resultado arroja una serie de valores que corresponden a una bloque de datos. Los valores arrojados por la función HASH son determinados por el parámetro HASHKEYS del comando CREATE CLUSTER, por ejemplo, supóngase que se le da un valor de 150 al parámetro HASHKEYS, significa que los valores clave del cluster oscilaran entre 0 y 149.

El número máximo de claves HASH asignadas por bloque de datos se determina por el parámetro SIZE del comando CREATE CLUSTER, éste parámetro contiene un valor estimado en bytes que se necesitará para almacenar los renglones asociados con cada valor clave del cluster.

Un cluster HASH almacena los renglones de datos consecutivamente en el mismo bloque.

La aplicación de la función HASH para almacenar o encontrar renglones de una tabla permite comprobar la utilidad de la teoría mencionada en el capítulo tres por parte de un sistema manejador de base de datos relacional específicamente ORACLE.

CONCLUSIONES

Durante el desarrollo del presente trabajo se hicieron patentes tres aspectos que resultan de estudiar el tema del almacenamiento físico, lógico y el acceso a los datos a través de un sistema manejador de bases de datos.

Primeramente, al analizar los diversos conceptos que conforman la teoría que sustenta el almacenamiento de los datos se hizo visible la vinculación de conceptos y aplicaciones impartidos en algunas materias en el transcurso de la carrera, tales como ***"Datos y Estructuras de Almacenamiento", "Estructuras de Datos", "Bases de Datos" y "Sistemas Operativos"***; ya que en cada una de estas se abordaron la gran mayoría de los conocimientos y conceptos que conforman la teoría que sustenta el proceso de almacenamiento de los datos. Entre dichos conceptos se encuentran las estructuras de datos (listas y árboles), el manejo de apuntadores y los métodos de acceso a los mismos, la gestión de memoria y procedimientos, entre otros.

El segundo aspecto, es la aplicación real de conceptos mencionados anteriormente, ya que al ejemplificar las estructuras de almacenamiento de un producto comercial específico, se confirmó la aplicación de gran parte de la teoría existente para llevar a cabo el proceso de almacenamiento de los datos a través del sistema manejador de bases de datos relacional **ORACLE**, aspecto que se hace presente al estudiar las estructuras de almacenamiento que maneja este **RDBMS** debido a que se aprecia claramente la similitud de conceptos que hay entre las estructuras propias del manejador y las estructuras teóricas presentadas en los primeros capítulos de este escrito; permitiendo así comprender la utilidad que se le da a la teoría existente en los actuales sistemas manejadores de bases de datos.

El tercer aspecto enfoca los beneficios que resultan de la buena administración de los componentes del proceso de almacenamiento de los datos (estructuras de datos, dispositivos de almacenamiento y técnicas de acceso a los datos); como principal resultado de esto se distingue el mejoramiento de una de las peticiones que con mayor frecuencia es demandada por parte de los usuarios finales en el manejo de la información: la rapidez en el acceso a los datos.

La unión de los tres aspectos antes mencionados y el desequilibrio evidente que existe entre la creación de aplicaciones que explotan los datos contenidos en una base de datos y la creación de sistemas manejadores de bases de datos permiten comprender la urgencia de crear software nuevo y de mejorar el ya existente. Si bien este trabajo solo cubre el aspecto del almacenamiento de los datos, detrás de los demás aspectos existe igualmente teoría que los sustenta, por tanto es posible crear sistemas manejadores de bases de datos funcionales y de gran calidad.

REFERENCIAS BIBLIOGRÁFICAS

[BEMA1994] Bertino, Eliza. Martino, Lorenzo. *Object-Oriented Database Systems. Concepts and Arquitecture*. USA, Addison-Wesley, 1994.

[DA]C1986] Date, Chris. *An introduction to database system.* Fifth edition. Volume I. USA, Addison-Wesley, 1990.

Enterprise DBA Part 1 A: Architecture and Administration. Volume 1. Instructor guide. USA, ORACLE Corporations, 1999.

Gorman, Michael M. *Database Management System.Understanding and Applying Database Technology.* USA, QED Information Sciencies Inc., 1991.

Greene, Joseph B. *ORACLE DBA. Survival guide.* First edition. USA, Sams Publishing, 1995.

Hansen, Gary W. y James V. *Diseño y Administración de bases de datos.* 2ª edición. México, Prentice Hall, 1997.

Introducción a ORACLE. Parte 1: Diseño Relacional de Bases de Datos. Guía del participante. México, ORACLE de México, 1994.

Krishnan, RaghRama. Gehrke, Johannes. *Database Management System*. Second edition USA, Mc Graw Hill, 2000.

[MA]A1977] Martin, James. *Organización de las bases de datos*.(Tr.: Adolfo Di Marco). México, Prentice Hall, 1977.

ORACLE 7 Server. Concepts Manual. USA, ORACLE Corporation, 1992.

ORACLE 9i SQL Reference, release 2 (9.2). USA, ORACLE Corporation, 2002.

[PAGW2001] Page, William G, jr. *ORACLE 8/8i. Edición especial.* México, Prentice Hall, 2001.

[PARU1997] Parrilla, Juan Carlos. Rubio, Juan José. *Sistemas Gestores de Bases de datos*. Madrid, Síntesis, 1997.

Pratt, Philip J. Adamski, Joseph J. *Database Systems. Management and Design.* USA, 1991.

Rolland, F. D. *Relational Database Management with ORACLE.* Second edition. USA, Addison-Wesley, 1994.

Stackwiak, Robert. Stern, Jonathan. *ORACLE Essentials*. First edition. USA, O'Reilly, 1999.

[TSHA1990] Tsai, Alice Y. H. *Sistemas de bases de datos. Administración y uso.* (Tr.: Alejandro Emilio Montes). México, Prentice Hall, 1990.

Ullman, Jeffrey D. Windom, Jennifer. *A first course in Database Systems*. USA, Prentice Hall, 1997.

[VOGO1991] Vossen, Gottfried. *Data Models, Database languages and database management systems.* USA, Addison- Wesley, 1991. (International Computer Science series).

Watson, Richard T. *Data Management. An organizational perspective.* USA, John Wiley & Sons, Inc., 1996.

DIRECCIONES ELECTRONICAS DE REFERENCIA

Instituto Tecnológico de Tepic. *Tutorial de la materia de administración de archivos.* Unidad VII, 2000.
http://www.geocities.com/seminarioittepic/unidadvii

La memoria en Oracle., 2001
http://www.zunda.net/cursos/oracle/memoria/tema02.htm

Márquez, Ma. Mercedes. *Apuntes de ficheros y Bases de datos.* Organización de ficheros y estructuras de acceso, 2001.
http://www3.uji.es/~mmarques/f47/apun/node1.html