Administración y Programación de Bases de Datos

Unidad 2: Diseño físico y seguridad en Bases de Datos

Gilberto Gutiérrez

DCCTI/UBB

Otoño 2017

Contenido

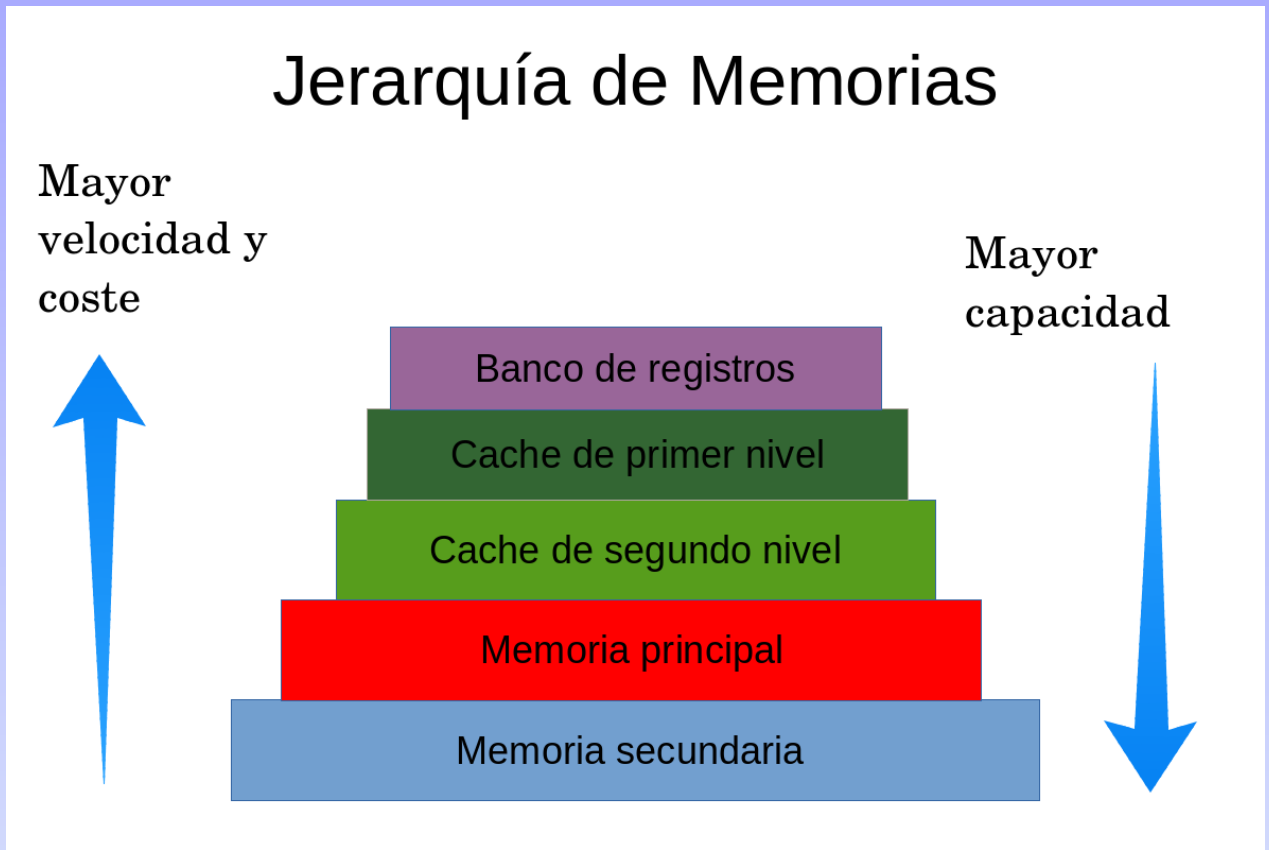


Figure 1: tomada de *http://www.apuntes.kosasguapas.com/2014/10/19/*

Introducción

Las Bases de Datos típicamente se almacenan en memoria secundaria por las siguientes razones:

- Tamaño de la Base de Datos.
- Persistencia de los datos
- Costo de almacenamiento. Memoria principal $>$ en un orden de magnitud que memoria secundaria por unidad de almacenamiento

Disco (Almacenamiento secundario)

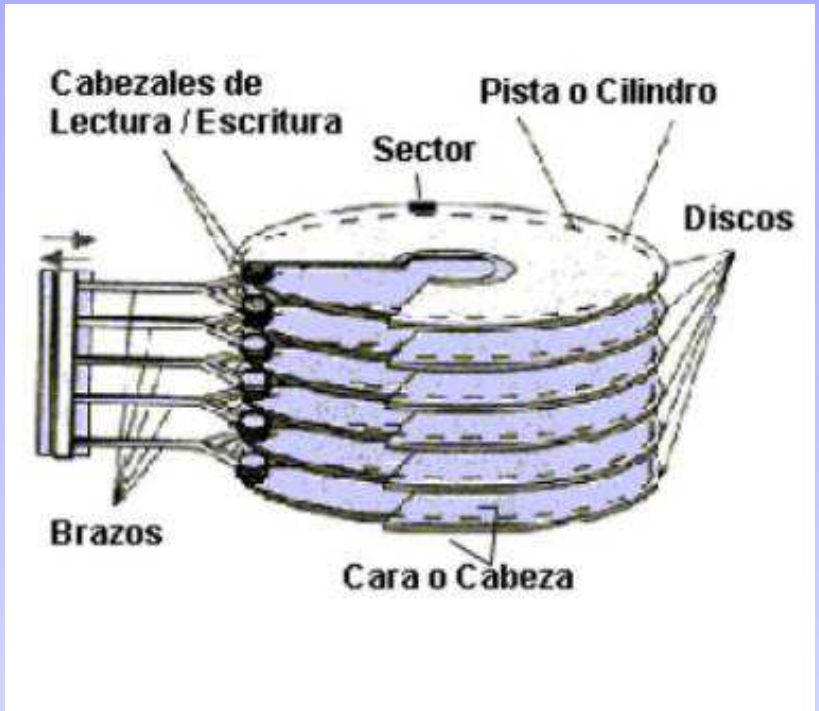


Figure 2: tomada de *https://osunal.wordpress.com/2013/05/16/estructura-fisica-y-logica-de-disco-discos-mecanicossan-nas-y-ssd/*

Bloque / Página

Unidad mínima de transferencia entre la memoria principal y el disco.

- Las pistas se dividen en bloques
- Los bloques se establecen por el S.O. al momento de inicializar (formatear) el disco.
- El tamaño del bloque se establece en la inicialización y no puede cambiarse.
- Tamaños de los bloques: 512 - 8.192 Bytes
- Los bloques están separados por información de control

Acceso a bloque de disco

- La dirección de hardware de un bloque: # Cilindro, # pista y # bloque dentro de la pista.
- LBA (Logical Block Addres: Dirección de bloque lógica. Número entre $0 \ldots n$.
- Al acceder a un bloque se debe proporcionar un buffer.
  - Para una lectura, el contenido del bloque se copia buffer.
  - Para escritura, el contenido del buffer se copia en el bloque de disco.

Tiempos

- Tiempo de búsqueda. Tiempo requerido para ubicar la cabeza en la pista correcta (3 a 8 ms, 2008 aproximadamente)
- Tiempo rotacional o de lantencia. Tiempo requerido para que el principio del bloque quede bajo la cabeza (2 ms)
- Tiempo de transferencia. Tiempo necesario para transferir los datos.

Tiempo de búsqueda y el tiempo rotacional son mucho mayores que el tiempo de transferencia

Registro y tipos de registros

Registro

Colección de valores o elementos de datos relacionados. Cada valor está formado por bytes y corresponde a un campo concreto del registro.

Tipos de registros

Colección de nombres de campos y sus correspondiente tipos de datos.

```
struct Amigos {
        int id;
        char nombre[20];
        int Fono;
        Date Fnac;
        char Sexo;
};
```

Registro y tipos de registros

- Registros de longitud fija
- Registros de longitud variable

Archivo

Secuencia de registros, generalmente del mismo tipo.

- Formado por registros de longitud fija
- Formado por registros de longitud variable
  - Campos de longitud variable (nombre de una persona)
  - Valores repetitivos de un campo (# telefónicos)
  - Campos opcionales
  - Diferentes tipos de registros

Factor de bloqueo

$$bfr = \left\lfloor \frac{B}{R} \right\rfloor,$$

con B el tamaño del bloque en bytes, R el tamaño del registro en bytes y $B \geq R$

espacio inutilizado

$B - (bfr * R)$ bytes

Organización extendida

Un registro puede abarcar más de un bloque

- Disminuir el espacio inutilizado
- $R > B$

Archivos

Asignación de bloques a disco

- Asignación contigua.
- Asignación enlazada. Cada bloque contiene un puntero al siguiente bloque.

Descriptor de archivo

Información del archivo que los programas necesitan conocer.

- Información para acceder a los bloques del archivo
- Tamaño de los campos
- Permisos
- Fecha de creación
- ...

Operaciones sobre archivos

1. Open()
2. Close()
3. Reset()
4. Find()
5. Insert()
6. Delete()
7. ...

Archivos

Organización

- Organización de los datos en registros
- Bloques y estructura de acceso
- Incluye la forma en que los registros y bloques se interconectan en el almacenamiento.

Métodos de acceso

Grupo de operaciones que se pueden aplicar al archivo para recuperar/actualizar la información.

Archivos Heap/Pila

- Se inserta un registro al final
- La eliminación es complicada.
  - Eliminación física
  - Eliminación lógica
- Búsqueda secuencial
- Se pueden usar índices secundarios para mejorar la búsqueda

Archivo Secuencial Ordenado

Los registros del archivo se encuentran ordenados físicamente en el disco en función de unos de sus campos.

- Campo de ordenación/clave
- Acceso secuencial/búsqueda binaria.
- Muy eficaz para búsqueda del tipo $<, \leq, >, \geq$
- La Inserción y la eliminación son operaciones costosas
- La actualización del campo clave también es costosa.

Archivos Secuenciales

Jerarquía de clases de flujo de JAVA

```
Object
   File
   FileDescriptor
   InputStream
      ByteArrayInputStream
      SequenceInputStream
      ...
      FileInputStream
      FilterInputStream
        DataInputStream
        ...
   OutputStream
      ByteArrayOutputStream
      SequenceOutputStream
      ...
      FileOutputStream
      FilterOutputStream
        DataOutputStream
        ...
   RandomAccessFile
```

Archivos Secuenciales

Algunas clases:

- File Permite obtener información de los archivos (fecha de creación, tamaño, etc.)
- InputStream/OutputStream Clases abstractas que definen métodos para realizar operaciones de entrada/salida
- FileInputStream/FileOutputStream Permiten realizar entrada/salida desde y hacia archivos.
- DataInputStream/DataOutputStream Permiten realizar entrada/Salida "formateada". Se pueden leer/grabar directamente objetos de tipo de datos primitivos.
- ...

Un ejemplo

```
import java.io.*;
 public class Amigo{
  int id;
  String nombre;
  int cel;
  String sexo;
Amigo() {}
Amigo(int elId, String elNombre, int elCel, String elSexo){
  id = elId; nombre = elNombre;
  cel = elCel; sexo = elSexo;
}
// Lee un amigo desde s
void readAmigo(DataInputStream s) throws IOException {
  id = s.readInt();  nombre= s.readUTF();
  cel = s.readInt(); sexo= s.readUTF();
 }
void writeAmigo(DataOutputStream s) throws IOException {
  s.writeInt(id);  s.writeUTF(nombre);
  s.writeInt(cel); s.writeUTF(sexo);
}
void Print() {
  System.out.println("Id: "+ id +" Nombre: "+ nombre + " Cel: " + cel + " Sexo: " + sexo);
}
}
```

Como escribir en un archivo secuencial (al final)

```
import java.io.*;
class writeAmigos {
public static void main (String args []) throws IOException {
   Scanner in = new Scanner(System.in);
   DataOutputStream out= new DataOutputStream(new FileOutputStream("amigos.dat",true));
   int elId, elCel;
   String elNombre, elSexo;
   Amigo a;
   for(;;){
      System.out.println("---- datos del amigo -----");
      System.out.print("ID (-1 --> termina) :");
      elId = in.nextInt();
      if(elId<0) break;
      System.out.print("Nombre :");
      elNombre = in.next();
      System.out.print("Celular : ");
      elCel = in.nextInt();
      System.out.print("Sexo (M/F): ");
      elSexo = in.next();
      a = new Amigo(elId,elNombre, elCel, elSexo);
      a.writeAmigo(out);
   }
  System.out.println("Terminado ... ");
  out.close();
 }
}
```

Como leer desde un archivo secuencial

```
  import java.util.Scanner;
  import java.io.*;
  class readAmigos {
  public static void main (String args []) throws IOException {
  DataInputStream in= new DataInputStream(new FileInputStream("amigos.dat"));
  Amigo a=new Amigo();
   try {
    for(;;) {
       a.readAmigo(in);
       a.Print();
      }
    } catch(EOFException e) { /*No hace nada */;}
  in.close();
 }
}
```

Problemas

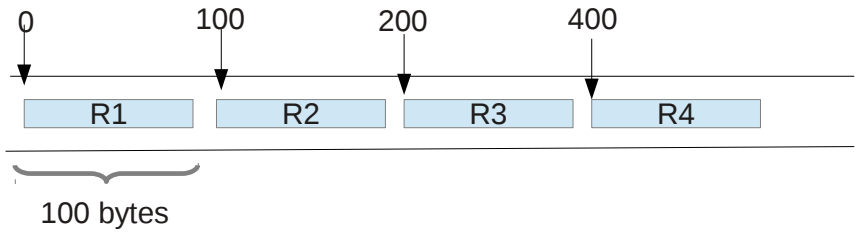
Para cada problema escriba un programa en JAVA.

1. Dado el **Id** de un amigo, muestre el nombre y fono del amigo almacenado en *amigo.dat*. Si no está en *amigo.dat* su programa debe emitir el mensaje `amigo con Id inexistente`.
2. Dado el **Id** de un amigo, búsquelo y elimínelo físicamente del archivo *amigo.dat*.
3. Suponga que el archivo *amigo.dat* es un archivo secuencial ordenado (con campo de ordenación el `Id`). Se desea insertar los datos de un nuevo amigo en el archivo *amigo.dat*. Escriba un programa JAVA que permita insertar el nuevo amigo de tal forma que el archivo resultante siga siendo secuencial ordenado.

Archivos de acceso aleatorio

- Acceso directo a un registro particular, sin necesidad de acceder a otros.
- Ideal registro de longitud fija. Es fácil calcular la posición relativa de un registro.
- Es fácil agregar un registro.
- Los datos almacenados se pueden actualizar o eliminar sin tener que reescribir todo el archivo.
- Podemos acceder a los registros tanto en forma aleatoria como secuencial.

Archivos de acceso aleatorio



Archivos de acceso aleatorio en JAVA

- `RandomAccessFile`. Tiene las capacidades de DataInputStream/DataOutputStream
- Los datos se leen/escriben a partir de la posición indicada por un puntero
- Todos los datos se leen/escriben como tipos de datos primitivos. Por ejemplo, al escribir un `int` se escriben 4 bytes en el archivo.
- ...

Algunos métodos

- RandomAccessFile(File file, String modo). Constructor
- RandomAccessFile(String name, String modo). Constructor
- char readChar(), int readInt(), ..., void writeInt(int), void write(byte[] b), ... Leer y escribir.
- void seek(long pos). Posiciona el puntero pos unidades desde el comienzo del archivo.
- long length(). Retorna la longitud del archivo
- void close(). Cierra el archivo.

Example

Clase Empleado

```
import java.io.*;
public class Empleado{
   int id;
   String nombre; //20 char
   String apellido; // 20 char
   byte depto;
   int sueldo;
 Empleado() {}
 Empleado(int elId, String elNombre, String elApellido, byte elDepto, int elSueldo){
   id = elId; nombre = elNombre; apellido = elApellido; depto = elDepto; sueldo = elSueldo;
 }
//Lee un empleado.
void readEmpleado(RandomAccessFile f) throws IOException { // Construye el amigo desde el archivo
   byte b1[] = new byte[20]; byte b2[] = new byte[20]; id = f.readInt();
   f.readFully(b1); nombre = new String(b1);  f.readFully(b2);
   apellido = new String(b2);depto = f.readByte(); sueldo = f.readInt();
 }
// Escribe en disco un empleado
void writeEmpleado(RandomAccessFile f) throws IOException {
   byte b1[] = new byte[20]; byte b2[] = new byte[20];
   f.writeInt(id); nombre.getBytes(0,nombre.length(),b1,0);
   f.write(b1); apellido.getBytes(0,apellido.length(),b2,0);
   f.write(b2); f.writeByte(depto); f.writeInt(sueldo);
 }
// Tamanho del registro empleado
 int size() { return 49;}
 void Print() { System.out.println(id +"\t"+ nombre + "\t" + apellido + "\t"+  depto + "\t" + sueldo);}
```

Example

Escribir un empleado

```
import java.util.Scanner;
import java.io.*;
class writeEmpleados {
public static void main (String args []) throws IOException {
   Scanner in = new Scanner(System.in);
   RandomAccessFile out= new RandomAccessFile("empleados.dat","rw");
   out.seek(out.length()); //al final del archivo
   int elId, elSueldo;
   byte elDepto;
   String elNombre, elApellido;
   Empleado e;
   for(;;){
      System.out.println("---- datos del Empleado -----");
      System.out.print("ID (-1 --> termina) :");
      elId = in.nextInt();
      if(elId<0) break;
      System.out.print("Nombre (20) :");
      elNombre = in.next();
      System.out.print("Apellido(20) :");
      elApellido = in.next();
      System.out.print("Depto : ");
      elDepto = in.nextByte();
      System.out.print("Sueldo : ");
      elSueldo = in.nextInt();
      e = new Empleado(elId,elNombre,elApellido, elDepto, elSueldo);
      e.writeEmpleado(out);
   }
```

Example

Leer secuencial empleado

```
import java.util.Scanner;
import java.io.*;
 class readEmpleados {
 public static void main (String args []) throws IOException {
 RandomAccessFile in= new RandomAccessFile("empleados.dat","r");
 Empleado em=new Empleado();
  try {
   for(;;) {
       em.readEmpleado(in);
       em.Print();
      }
    } catch(EOFException ex) { /*No hace nada */;}
  in.close();
 }
}
```

Example

Busca un empleado

```
import java.io.*;
class searchEmpleados {
public static void main (String args []) throws IOException {
   if(args.length !=1) { System.out.println("mode de uso: searchEmpleado id"); System.exit(1);}
   int id   = new Integer(args[0]).intValue();
   Empleado e = new Empleado();
   RandomAccessFile file= new RandomAccessFile("empleados.dat","r");
   long end = file.length();
   long pos=  (long) (id-1)*e.size();
   if(pos >= end) {System.out.println("No existe empleado !!"); System.exit(1);}
   file.seek(pos);
   e.readEmpleado(file);
   e.Print();
  file.close();
 }
}
```

Example

Actualiza un empleado

```
import java.io.*;
class updateEmpleados {
public static void main (String args []) throws IOException {
   if(args.length !=2) { System.out.println("mode de uso: searchEmpleado id NuevoNombre"); System.exit(1);}
   int id   = new Integer(args[0]).intValue();
   Empleado e = new Empleado();
   RandomAccessFile file= new RandomAccessFile("empleados.dat","rw");
   long end = file.length();
   long pos=  (long) (id-1)*e.size();
   if(pos >= end) {System.out.println("No existe empleado con id " + id); System.exit(1);}
   Empleado em = new Empleado();
   em.nombre = args[1];
   file.seek(pos);
   em.readEmpleado(file);
   file.seek(pos);
   em.writeEmpleado(file);
  file.close();
 }
}
```

Archivos

Problemas:

1. Dados dos Id de empleados, Id_1 e id_2, escriba un program JAVA para listar los nombres de los empleados cuyo *id* se encuentra entre Id_1 e Id_2 (incluidos).
2. Sean los siguientes atributos para los Departamentos (Depto): $depto\{IdDepto, Nombre\ char(20), IdJefe\}$ escribir programas JAVA para: (i) Insertar de manera ordenada por el *IdDepto* en un archivo llamamado *depto.dat* y otro programa para (ii) listar todos los departamentos en *depto.dat*.
3. Escriba un programa JAVA para listar el nombre de cada empleado y el nombre del departamento al cual se encuentra asignado.

Factores que influyen en el diseño físico

A. Análisis de las consultas y las transacciones de la base de datos

Por cada consulta especificar:

1. Las tablas a las que accede cada consulta
2. Atributos especificados en la condición de selección
3. Condición de igualdad o de rango
4. Atributos con los cuales se harán los join
5. Atributos a recuperar por la consulta

Operaciones de actualización

1. Tablas que actualizarán
2. Tipo de operación (inserción, modificación, eliminación)
3. Atributos con los que se especifican las condiciones de selección para una eliminación o actualización
4. Que atributos cambiarán a causa de una actualización

B. Análisis de la frecuencia de ejecución esperada de consultas y transacciones

Sobre todas las transacciones:

- Frecuencia de uso esperada de cada atributo en selección
- Frecuencia de uso esperada de atributos en Join
- ...

Regla 80-20

Aproximadamente el 80% del procesamiento se debe al 20% de las consultas y transacciones.

C. Análisis de las restricciones de tiempo para consultas y transacciones

Restricciones "duras". Ejemplo: Una transacción debe terminar en 5 segundos en el 95% de los casos y nunca tardar más de 20 segundos.

Los atributos usados en la selección que las consultas y transacciones utilizan con restricciones de tiempo se convierten en candidatos a formar un índice.

D. Análisis de las frecuencias esperadas de las operaciones de actualización

Cantidad de índices de tal manera de no hacer lento las operaciones de actiualización de una tabla.

E. Análisis de las restricciones de unicidad en los atributos

Índices para los atributos (o conjunto de atributos) claves.

Decisiones sobre el diseño físico

Decisiones de diseño sobre la indexación

1. Cuándo indexar un atributo
2. Qué atributos indexar. Atributos implicados en consultas frecuentes
3. Cuándo configurar un índice de agrupación. Los índices de agrupación mejoran las búsquedas por rango.
4. Arbol B^+ o Hashing

Decisiones sobre el diseño físico

Desnormalización para agilizar las consultas

- Sacrificar los "ideales de la normalización" en favor de la ejecución rápida de las consultas y transacciones más frecuentes.
- Se añaden a una tabla atributos que permiten responder una consulta y evitar un join.
- Se produce redundancia de información
- Peligro de inconsistencia de la base de datos.

Decisiones sobre el diseño físico

Desnormalización

Sea la tabla (1FN)
ASSIGN(idEmp, idProy, NombreEmp, TituloTrabaEmp, PorcentajeAsignado,NombreProy, IdDirectorProy, NombreDirectorProy)
y las DFs
$idProy \longrightarrow NombreProy, IdDirectorProy$
$IdDirectorProy \longrightarrow NombreDirectorProy$
$idEmp \longrightarrow NombreEmp, TituloTrabajEmp$

```
EMP(idEmp, NombreEmp, TituloTrabajEmp)
PROY(idProy, NombreProy, IdDirectorProy)
EMP_PROY(IdEMp, idProy, PorcentajeAsignado)
```

$EMP\_PROY * EMP * PROY \bowtie_{idDirectorProy=IdEmp} EMP$

Refinamiento de una Base de Datos Relacional

Revisitar decisiones de diseño visto hasta ahora

Objetivos de la refinación

- Conseguir que las aplicaciones se ejecuten más rápidamente
- Reducir los tiempos de respuestas de las consultas y transacciones
- Mejorar el rendimiento global de las transacciones

Refinamiento de una Base de Datos Relacional

Estadísticas

- Tamaño de las tablas individuales
- Número de valores distintos en una columna
- Frecuencia de ejecución de una consulta o transacción
- ...

Refinamiento de una Base de Datos Relacional

Refinamiento de los índices

- Ciertas consultas tardan demasiado por carecer de un índice
- Índices muy volátiles. Mucha actualización
- Reorganización de los índices (muchas eliminaciones).
- ...

Refinamiento de una Base de Datos Relacional

Refinamiento del diseño de la Base de Datos

Los cambios se deben reflejar en el diseño lógico y físico. Los cambios pueden ser de la siguiente naturaleza:

- Desnormalización
- Para el conjunto de tablas puede haber opciones de diseño (3FN, BCFN)
- División vertical de tablas.
- Incorporar redundancia de datos (repetir columnas)
- División horizontal

Refinamiento de una Base de Datos Relacional

Refinación de consultas

Algunas situaciones que motivan refinar una consulta:

- No se utiliza un índice ante expresiones aritméticas (Salario /360 > 10.50)
- En consultas anidadas con IN no se utiliza índice.
  ```
  SELECT Dni FROM EMPLEADO WHERE Dno IN (SELECT
  NumeroDepto FROM DEPARTAMENTO WHERE DniDirector =
  33333);
  ```
- Evitar DISTINCT (provoca ordenar la salida y a veces no es necesario).
- No abusar del uso de tablas temporales en las consultas

Refinación de consultas (cont.)

- Evitar consultas como la siguiente:

```
SELECT Dni FROM EMPLEADOS E
WHERE sueldo= (SELECT MAX(Sueldo) FROM EMPLEADO AS M
                WHERE M.Dno = E.Dno);
SELECT MAX(Sueldo) AS SueldoMax, Dno INTO TEMP FROM EMPLEADO GROUP
BY Dno;
SELECT Dni FROM EMPLEADO, TMP WHERE Sueldo= SueldoMax and
EMPLEADO.Dno = TEMP.Dno
```

- El orden de las tablas en FROM puede incidir en el tiempo de ejecución.

Refinación de consultas (cont.)

Algunos optimizadores funcionan peor con consultas anidadas que con las no anidadas.

- Consultas anidadas
  - Subconsultas no correlacionadas con funciones agregadas en la consulta interna
  - Subconsultas no correlacionadas sin funciones de agregación
  - Subconsultas correlacionadas con funciones de agregación en la consulta interna
  - Subconsultas correlacionadas sin funciones de agregación
- A veces es conveniente no usar una vista. Es mejor usar directamente las tablas bases.

Refinación de consultas/Directricies adicionales

- Usar Unión en lugar de OR en la clausula WHERE. Ejemplo la consulta

```
SELECT Nombre, Apellidos, Sueldo, Edad FROM EMPLEADO
WHERE Edad > 45 OR Sueldo < 50000;
```

Se podría reemplazar por la consulta

```
SELECT Nombre, Apellidos, Sueldo, Edad FROM EMPLEADO
WHERE Edad> 45
UNION
SELECT Nombre, Apellidos, Sueldo, Edad FROM EMPLEADO
WHERE Sueldo < 50000
```

donde se podrían usar los índices de los atributos `Edad` y `Sueldo`

Refinación de consultas/Directricies adicionales (cont.)

- Transformaciones para acelerar una consulta
  - Transformar NOT en una expresión positiva
  - Reemplazar los operadores IN, =ALL, =ANY e =SOME por JOIN
  - EquiJoin con selección. Establecer la condición de la selección en ambas tablas.

Tipos de Seguridad

La seguridad es un tema muy amplio que comprende muchos conceptos. Algunos son:

- Aspectos Legales. Ciertos tipos de datos son privados
- Políticas de estado, de empresas, etc. Créditos de clientes o fichas clínicas de los pacientes
- Relativos al sistema. Ejemplo, niveles del sistema en que debería reforzar las distintas funciones de seguridad (nivel físico del hardware, a nivel del S.O o a nivel del SABD)
- Diferentes niveles de seguridad dentro de la organización. No clasificado, secreto, alto secreto, etc.

Amenazas a las Bases de Datos

Tienen como consecuencia:

- Pérdida de integridad. Protección contra modificación no autorizada.
- Pérdida de disponibilidad. Los objetos de la BD deben estar disponibles para un usuario o programa de aplicación
- Pérdida de confidencialidad. Protección de los datos frente a accesos no autorizado.

Medidas de Control

- Control de accesos. Cuentas y contraseña
- Control de inferencias. Bases de datos estadísticas
- Control de flujo. Ejemplos. Evitar que un programa guarde los datos confidenciales de los clientes. Bloquear la transmisión de datos militares a un usuario desconocido.
- Cifrado de datos. Codificación de partes la base de datos (Número de tarjetas de crédito)

El DBA y la seguridad

Responsable de la seguridad general del sistema de bases de datos. Puede ejecutar los siguientes tipos de acciones:

- Creación de cuentas.
- Concesión de privilegios.
- Revocación de privilegios.
- Asignación del nivel de seguridad.

Protección de accesos, cuentas de usuario y auditorías

- DBA crea cuenta y contraseña
- Bitácora con registro de accesos
- Auditorías

Concesión y revocación de privilegios

Niveles de asignación de privilegios:

- El nivel de cuenta. El DBA especifíca los privilegios en particular que posee cada cuenta, independiente de las relaciones en la base de datos.

```
GRANT CREATE TABLE, CREATE VIEW, CREATE TRIGGER TO ana, jose;
```

- El nivel de relación. El propietario de la relación/vista cede o revoca los privilegios a otros usuarios.
  - Privilegio de selección
  - Privilegio de modificación. Se puede especificar a nivel relación o de atributos
  - Privilegios de referencia. Permite establecer restricciones de integridad

```
GRANT INSERT ON EMPLEADO, DEPARTAMENTO TO ana, jose
```

Especificación de privilegios mediante vistas

Las vistas en si mismo son mecanismos de autorización discrecional.

Example

- Si el propietario A de una relación R quiere que un usuario B pueda selecionar solo algunos campos de R, entonces A puede crear una vista V de R que incluye esos atributos y después autorizar SELECT sobre V a B.
- Similar se puede hacer para que B solo vea algunas tuplas de R.

Revocación de privilegios

Quitar privilegios previamente concedidos.

Example

```
REVOKE INSERT ON EMPLEADO FROM ana
```

Propagación de privilegios

Cesión de privilegios con opción de cederlos a otros usuarios

Example

El dueño *A* de la tabla `EMPLEADO` concede privilegio de SELECT a ana

```
GRANT SELECT ON EMPLEADO TO ana WITH GRANT OPTION
```

Ahora ana puede conceder el privilegio a otro usuario

```
GRANT SELECT ON EMPLEADO TO jose
```

¿ Que ocurre con jose si *A* revoca el privilegio SELECT sobre EMPLEADO al usuario ana ?

Example

Suponganos que el DBA crea 4 cuentas $A1, A2, A3, A4$.

- `GRANT CREATE TABLE A1;` (privilegio a nivel de cuenta)
- Supongamos que $A1$ crea las tablas EMPLEADO y DEPARTAMENTO. $A1$ tiene todos los privilegios de relación sobre ambas relaciones.
- `GRANT INSERT, DELETE ON EMPLEADO, DEPARTAMENTO to A2;`
- `GRANT SELECT ON EMPLEADO, DEPARTAMENTO to A3 WITH GRANT OPTION;`.
- El usuario $A3$ puede conceder privilegios.
  `GRANT SELECT ON EMPLEADO to A4;`
- $A1$ decide revocar el privilegio cedido a $A3$;
  `REVOKE SELECT ON EMPLEADO FROM A3;`

Example

Suponganos que *A*1 desea ceder permisos limitados a *A*3 sobre la relación EMPLEADO y desea permitir que *A*3 propague el privilegio. Se desea que sólo se puedan ver los atributos *Nombre*, *fechaNac* y *Direccion* y solo para las tuplas con *DNo* igual a 5.

```
CREATE VIEW EMP5 AS
SELECT Nombre, FechaNac, Direccion
FROM EMPLEADO
WHERE Dno=5
GRANT SELECT ON EMP5 to A3 WITH GRANT OPTION
```

Control de acceso discrecional

Example

Suponganos que $A1$ desea que $A4$ actualice solamente el atributo *Sueldo* de la relación EMPLEADO. $A1$ podrá ejecutar el siguiente comando:

```
GRANT UPDATE ON EMPLEADO(sueldo) TO A4
```

Los privilegios UPDATE e INSERT pueden especificar atributos. SELECT y DELETE no lo permiten. Se pueden especificar mediante vistas.

Control de acceso basado en roles

La idea es "Los permisos se asocian a ROLES y a los usuarios se les asignan los roles apropiados"
Con GRANT y REVOKE se conceden y revocan privilegios a los roles.
CREATE ROLE
DROP ROLE
Un usuario puede tener varios roles.

Example

```
SQL> create role curso;

Role created.

SQL> grant curso to ggutierr;

Grant succeeded.

SQL> grant select on amigos to curso;

Grant succeeded.

SQL>
```