EDAT

1. **INTRO**

Una base de datos es un conjunto de tecnologías que permite almacenamiento de datos estructurados, su versión más simple son las tablas relacionadas. Estas tablas se almacenan de manera persistente en memoria secundaria (campos y registros). La representación en tablas proporciona mucha generalidad a la representación de los datos. Las tablas se relacionan mediante una serie de campos especiales que son lo que se denominan básicamente claves primarias y externas. Una BD se puede crear, acceder y actualizar.

Base de datos = conjunto de tablas, esquemas, restricciones… Tabla = registros y campos SQL = lenguaje de creación, actualización y consulta de bases de datos SGBD = software de gestión y acceso a bases de datos

1. **SQL**

Elementos fundamentales de una base de datos en SQL:

Base de datos = conjunto de tablas RELACIONADAS

Tabla (relación, entidad, esquema…) =

– Estructura fija de campos (esquema)

– Conjunto de registros con valores de campos

Campo (atributo, propiedad, “columna”), tiene un tipo de dato

Registro (tupla, “fila”)

Clave primaria

Clave secundaria

Clave externa

Tipos de datos:

Numéricos:

–Entero de distintos tamaños (INTERGER o INT, SMALLINT)

–Reales de distinta precisión (FLOAT o REAL, DOUBLE PRECISION)

–Números con formato (NUMERIC(t,d), DECIMAL(t,d), d menor t, d=0 por defecto, t=n dig, d= n dig a la derecha del punto)

Cadenas de caracteres:

–CHAR (n) – longitud fija con relleno a blancos

–CHAR VARYING (n) - longitud variable con límite

–TEXT - longitud variable sin límite

Cadenas de bits:

–BIT(n)

–BIT VARYING (n)

Fecha y Hora:

–DATE (10 posiciones), YYYY-MM-DD ´

–TIME (8 posiciones), HH:MM:MM

–TIME (i), carácter separador más i posiciones para fracciones de segundo

–WITH TIME ZONE, 6 posiciones extra para el desplazamiento respecto al uso horario estándar universal, +-HH.MM

–TIMESTAMP YYYY-MM-DD HH.MM.SS. fracciones de segundo (6 posiciones), es opcional el calificador WITH TIME ZONE

–INTERVAL, periodo de tiempo (están cualificado para ser de dos tipos de intervalos generalmente)

•AÑO/MES

•DIA/HORA

Esquema:

**CREATE SCHEMA** EMPRESA **AUTHORIZATION** nombre;

Dominio:

**CREATE** DOMAIN nombre **[AS]** tipo\_datos

**[Definición\_por\_defecto]**

**[Restricciones]**

Definición\_por\_defecto

**DEFAULT** {literal|función|NULL}

Restricciones

[**CONSTRAINT** nombre]

CHECK (expresión\_condicional)

Ejemplos:

**CREATE** DOMAIN CIUDADES CHAR(15)

**DEFAULT** ´Madrid´

**CONSTRAINT** MirestricciondeCiudades

CHECK (VALUE IN(‘Atenas’, ‘Dublin’,……., ‘Madrid’))

**CREATE** DOMAIN NumEmp NUMERIC(4)

**DEFAULT** 0

CHECK (VALUE IN NOT NULL)

Tablas:

**CREATE** TABLE nombre (

campo1 tipo1 [restricciones1],

campo2 tipo2 [restricciones2],

…,

[restricciones]

);

**ALTER** TABLE nombre ADD COLUMN campo tipo [restricciones];

**ALTER** TABLE nombre ADD restricción;

**ALTER** TABLE nombre DROP COLUMN campo;

**DROP** TABLE nombre;

**DROP** CONSTRAINT nombre-restricción;

Ejemplos:

**CREATE** TABLE Contacto (

usuario1 varchar(30) REFERENCES Usuario (nick),

usuario2 varchar(30) REFERENCES Usuario (nick),

PRIMARY KEY (usuario1,usuario2)

);

**CREATE** TABLE Usuario (

nick varchar(30) PRIMARY KEY,

nombre text NOT NULL,

email text NOT NULL UNIQUE

);

**CREATE** TABLE Escucha (

usuario varchar(30) ,

cancion int REFERENCES Cancion (id),

instante timestamp,

PRIMARY KEY (usuario,cancion,instante)

);

**ALTER** TABLE Escucha ADD instante timestamp;

**ALTER** TABLE Escucha DROP COLUMN instante;

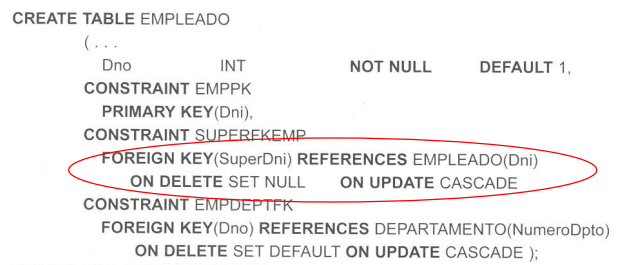
**ALTER** TABLE Escucha ADD FOREIGN KEY (usuario)

REFERENCES Usuario (nick);

**ALTER** TABLE Usuario ADD PRIMARY KEY (nick);

– ON DELETE CASCADE: elimina las tuplas referenciadas en cascada cuando eliminamos algo de la clave externa.

– ON UPDATE CASCADE: Actualiza la clave externa por la que se ha cambiado, en cascada



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Tabla

Descripción generada automáticamenteTabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Para añadir o eliminar tablas, atributos, restricciones:

**DROP SCHEMA** empresa **CASCADE**; Se elimina todo el esquema con todos los elementos (…. empresa **RESTRICT** solo se elimina si no contiene elementos).

**DROP TABLE** subordinado **CASCADE**; (se elimina la relación y su definición, **RESTRICT** solo se elimina la tabla si no hace referencia a otra tabla).

**DELETE FROM** tabla [WHERE …]; Para eliminar tuplas (**DELETE FROM** empleado WHERE ‘dni=123456789’; **DELETE FROM** EMPLEADO **WHERE** Apellido1=‘Cabrera’;).

**ALTER TABLA** EMPRESA.EMPLEADO **ADD COLUMN** Trabajo VARCHAR (12); (no se introduce un valor para la nueva columna, se pone toda la nueva columna a NULL).

**ALTER TABLA** EMPRESA.EMPLEADO **DROP COLUMN** dirección **CASCADE**; (**RESTRICT** solo se elimina la columna si no hay vistas o restricciones que hagan referencia a esta columna).

**ALTER TABLA** EMPRESA.DEPARTAMENTO **ALTER COLUMN** DniDirector **DROP DEFAULT** (se elimina la clausula predeterminada para DniDirector)

**ALTER TABLA** EMPRESA.DEPARTAMENTO **ALTER COLUMN** DniDirector **SET DEFAULT** ‘333445555’(se define la clausula predeterminada para DniDirector)

**ALTER TABLA** EMPRESA.EMPLEADO **DROP CONSTRAINT** SUPERFKEMP **CASCADE**; (se elimina la restricción SUPERFKEMP de la relación EMPLEADO)

**INSERT INTO** EMPLEADO **VALUES** (‘Ricardo’, Roca’,’Flores’,’653298653’,’30-12-1962’,’Los Jarales, 47’,’H’,’37000’, ‘653298653’,4); (añade una nueva tupla, al menos hay que añadir los que están explícitamente en la definición de tabla a NOT NULL );

**INSERT INTO** EMPLEADO (Nombre , Apellido1, Dno, Dni) **VALUES** (‘Ricardo’, ‘Roca’,’4’,’653298653’); (añade parte de una nueva tupla, los no especificados se establecen a DEFAULT o a NULL);

**UPDATE** PROYECTO UbicaciónProyecto=‘Valencia’, NumDptoProyecto = 5

**WHERE** NumProyecto=10;

**UPDATE** Empleado **SET** Sueldo=Sueldo\*1.1

**WHERE** Dno **IN** (**SELECT** NumeroDpto **FROM** DEPARTAMENTO

**WHERE** NombreDpto=‘Investigación’);

**IN** vamos a utilizarlo en consultas anidadas también.

**TRUNCATE** tabla; (quita todas las filas de una tabla, pero permanecen la estructura y sus columnas, las restricciones, los índices, etc. Para quitar la definición de tabla además de los datos: DROP TABLE)

**BEGIN**;

Secuencia de comandos que alteran una BD determinada

**COMMIT**; (este comando finaliza las transacciones haciendo los cambios permanentes y visibles a todos los usuarios)

**BEGIN** inicia un bloque de transacción, es decir, todas las declaraciones después de un comando **BEGIN** se ejecutarán en una sola transacción hasta que de manera explícita haya un **COMMIT** (plasma todos los cambios en la BD) o un **ROLLBACK** (deshace todos los cambios).

Con el comando **SAVEPOINT**, también puedes grabar puntos interesantes intermedios.

Consultas:

**SELECT** campos

**FROM** tablas

**WHERE** condición;

Ejemplos:

SELECT "FechaNac", "Direccion"

FROM "EMPLEADO"

WHERE "Nombre"='Jose' AND "Apellido1"='Perez' AND "Apellido2"='Perez';

SELECT "EMPLEADO"."Nombre", "EMPLEADO"."Apellido1", "EMPLEADO"."Direccion"

FROM "EMPLEADO", "DEPARTAMENTO"

WHERE "EMPLEADO"."Dno"="DEPARTAMENTO"."NumeroDpto" AND "DEPARTAMENTO"."NombreDpto"='Investigacion';

Concatenación con JOIN:

SELECT Nombre, Apellido1, Dirección

FROM (EMPLEADO JOIN DEPARTAMENTO ON Dno= NumeroDpto)

WHERE NombreDpto=‘Investigación’;

Tipos de JOIN:

INNER Por defecto (no hace falta ponerlo).

NATURAL La condición consiste en igualdad entre la combinación de los campos que se llamen igual entre ambas tablas (EQUIJOIN) (no se repiten los campos).

LEFT | RIGHT | FULL Se añaden también filas que no cumplen la condición (incompatible con INNER)(OUTER JOIN).

– R (->< Dni=DniDirector) S (LEFT OUTER JOIN, mantiene cada tupla de la relación izquierda aunque no se encuentre ninguna tupla en S que cumple la conexión, esos atributos se rellenan a NULL).

– R (><- Dni=DniDirector) S (RIGHT OUTER JOIN, mantiene cada tupla de la relación derecha aunque no se encuentre ninguna tupla en R que cumple la conexión, esos atributos se rellenan a NULL).

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Consultas anidadas:

SELECT campos

FROM tabla

WHERE campo1, campo2, … IN (SELECT campo1, campo2, …);

Ejemplos:

SELECT DISTINCT UbicacionProyecto

FROM PROYECTO

WHERE NumProyecto IN

(SELECT NumProyecto

FROM PROYECTO, DEPARTAMENTO, EMPLEADO

WHERE NumDptoProyecto =NumeroDpto AND DniDirector=Dni AND (Apellido1=‘Pérez’ OR Apellido1=‘Ochoa’;))

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Opciones de consultas anidadas:

SELECT campos

FROM tabla

WHERE campo comparación (SOME | ALL) (SELECT …);

SELECT campos

FROM tabla

WHERE EXISTS (SELECT …);

SELECT campos

FROM tabla

WHERE (SELECT …) CONTAINS (SOME | ALL) (SELECT …);

Ejemplos:

SELECT u2.nombre

FROM Usuario AS u1, Usuario as u2 ´

WHERE

(u1.nick, u2.nick) IN

((SELECT usuario1, usuario2 FROM Contacto)

UNION

(SELECT usuario2, usuario1 FROM Contacto))

AND u1.nombre = 'Rosario';

Enumere el nombre de todos los empleados que trabajan en algún proyecto controlado por el departamento 5. El segundo SELECT me proporciona los números de proyecto que controla el departamento 5:

SELECT DISTINCT "EMPLEADO"."Nombre"

FROM "TRABAJA\_EN", "EMPLEADO"

WHERE "EMPLEADO"."Dni"="TRABAJA\_EN"."DniEmpleado" AND "TRABAJA\_EN"."NumProy" IN (SELECT "PROYECTO"."NumProyecto"

FROM "PROYECTO"

WHERE "PROYECTO"."NumDptoProyecto"='5');

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Algebra de conjuntos:

(SELECT usuario2 FROM Contacto WHERE usuario1 = 'charo'

UNION

SELECT usuario1 FROM Contacto WHERE usuario2 = 'charo')

INTERSECT

(SELECT usuario2 FROM Contacto WHERE usuario1 = 'lola'

UNION

SELECT usuario1 FROM Contacto WHERE usuario2 = 'lola')

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Ejemplos:

(Obtener una lista de los números de los proyectos que impliquen a cualquier empleado cuyo primer apellido sea ‘Campos’, independientemente de que sean trabajadores o directores del departamento que gestiona dicho proyecto): También se puede hacer con UNION:

SELECT "PROYECTO"."NumProyecto"

FROM "PROYECTO"

WHERE "PROYECTO"."NumProyecto" IN

((SELECT "PROYECTO"."NumProyecto"

FROM "PROYECTO", "DEPARTAMENTO", "EMPLEADO"

WHERE "PROYECTO"."NumDptoProyecto"="DEPARTAMENTO"."NumeroDpto" AND "DEPARTAMENTO"."DniDirector"="EMPLEADO"."Dni" AND "Apellido1"='Campos')

UNION

(SELECT "TRABAJA\_EN"."NumProy"

FROM "TRABAJA\_EN", "EMPLEADO"

WHERE "TRABAJA\_EN"."DniEmpleado"="EMPLEADO"."Dni" AND "Apellido1"='Campos'));

(Todos los empleados que NO trabajen en el proyecto 2, con la operación resta algebraica, EXCEPT)

SELECT "EMPLEADO"."Nombre", "EMPLEADO"."Apellido1", "EMPLEADO"."Apellido2"

FROM "EMPLEADO", "TRABAJA\_EN"

WHERE "EMPLEADO"."Dni"="TRABAJA\_EN"."DniEmpleado" EXCEPT

(SELECT "EMPLEADO"."Nombre", "EMPLEADO"."Apellido1", "EMPLEADO"."Apellido2"

FROM "EMPLEADO", "TRABAJA\_EN"

WHERE "EMPLEADO"."Dni"="TRABAJA\_EN"."DniEmpleado" AND "TRABAJA\_EN"."NumProy"='2');

Orden y agregación:

SELECT COUNT (campos) FROM tabla … [GROUP BY campo1, campo2, …]; SELECT SUM | MAX | MIN | AVG (campo) FROM tabla … [GROUP BY campo1, campo2, …]; SELECT … [ORDER BY campo1, campo2, …];

Ejemplos:

SELECT COUNT (\*) FROM Escucha JOIN Cancion ON cancion = id

WHERE titulo = 'Norwegian Wood';

SELECT autor, COUNT (\*) FROM Escucha JOIN Cancion ON cancion = id

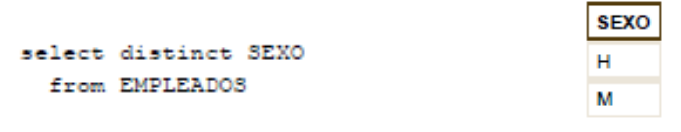
GROUP BY autor;

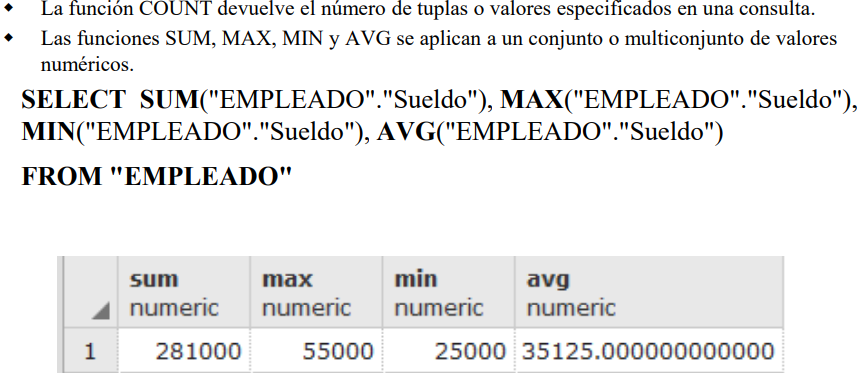
SELECT \* FROM Usuario

WHERE (SELECT COUNT (\*) FROM Contacto

WHERE usuario1 = nick OR usuario2 = nick) > 2;







Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Filtro HAVING COUNT:

Diagrama, Texto

Descripción generada automáticamente

Vistas:

Dan un nombre a una consulta, permiten usarla como tabla Útil para reutilizar consultas y evitar ejecutarlas varias veces Pueden configurarse para que se almacenen en disco

Ejemplos:

CREATE VIEW Contactos\_Usuario AS

SELECT u1.nick, u2.nombre FROM Usuario AS u1, Usuario as u2

WHERE (u1.nick, u2.nick) IN

((SELECT usuario1, usuario2 FROM Contacto)

UNION (SELECT usuario2, usuario1 FROM Contacto));

SELECT nombre FROM Contactos\_Usuario WHERE nick = 'pepe';

CREATE VIEW "TRABAJA\_EN1"

AS SELECT "Nombre" , "Apellido1" , "NombreProyecto" , "Horas"

FROM "EMPLEADO" , "PROYECTO" , "TRABAJA\_EN"

WHERE "Dni"="DniEmpleado" AND "NumProy"="NumProyecto";



1. **Modelo entidad relación**

Entidad: Objeto del mundo real con existencia independiente

• Físico: persona, coche, casa, empleado etc.

• Conceptual: puesto de trabajo, etc.

Relación: Es un vínculo que nos permite definir una dependencia entre varias entidades (ahora las vemos en detalle).

Atributos: Las diferentes propiedades que pueden tener la entidades.

• Atómicos o compuestos

• Monoevaluados o multievaluados

• Almacenados o derivados

• NULL

•Complejos

Claves: conjunto de atributos con ciertas propiedades de unicidad para las entidades:

• Superclave:

• Conjunto de atributos cuya combinación es única para un tipo de entidad (por ejemplo, el conjunto total de atributos de un tipo de entidad es una superclave, esto es trivial).

• Ejemplos: Dni + FechaNac es superclave de EMPLEADO.

• Clave:

• Una superclave mínima, también llamada clave candidata (Equivale a UNIQUE en SQL), pueden existir varias claves mínimas (claves candidatas de la entidad): por ejemplo en BD Música que vimos en SQL, nick es clave y email es también clave de la entidad Usuario.

• Ejemplos: Dni + FechaNac no es clave para EMPLEADO, en cambio Dni si es clave ya que es mínima.

• Claves: conjunto de atributos con ciertas propiedades de unicidad para las entidades:

• Clave primaria:

• Una clave que se designa como primaria para un tipo de entidad (si hay varias candidatas se elige una de ellas).

• Es la se utiliza para indexar a nivel físco (lo veremos más adelante…).

• Equivale a PRIMARY KEY en SQL.

• La elección entre claves candidatas es arbitraria.

• Notación gráfica en el modelo E-R: subrayado.

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Imagen que contiene Diagrama

Descripción generada automáticamente

**UNO a UNO:** 1:1 -> Una entidad de A se relaciona únicamente con una entidad en B y viceversa.

**UNO a VARIOS:** 1:N -> Una entidad en A se relaciona exclusivamente con una entidad en B. Pero una entidad en B se puede relacionar con 0 o muchas entidades en A.

**VARIOS a UNO:** N:1 -> Una entidad en A se relaciona con cero o muchas entidades en B. Pero una entidad en B se relaciona con una única entidad en A.

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente**VARIOS a VARIOS:** N:M -> Una entidad en A se puede relacionar con 0 o muchas entidades en B y viceversa.

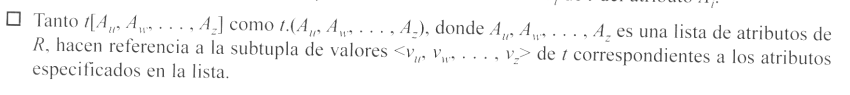
Diagrama

Descripción generada automáticamente

1. **Modelo relacional**

Texto, Carta

Descripción generada automáticamente



• El ESQUEMA de una relación sirve para describir la relación: R(A1, …, An)

• El GRADO de una relación es el número de atributos de su esquema.

• La RELACIÓN es la tabla de tuplas.

• Valores Nulos: se diseña el modelo para evitar esto

• Superclave: conjunto de atributos que identifican de manera única cada fila de la relación.

• Clave: conjunto de atributos mínimos que identifican de manera única cada fila de la relación.

• Clave primaria: la que se escoge de las claves candidatas

• Clave externa: conjunto de atributos de una relación que forman parte de la clave de otra relación.

• Las transformaciones generales son:

– Entidad -> Esquema relacional (tabla de tuplas) con la elección de una C-1º

• Atributos atómicos -> Columna de la tabla

• Atributos compuestos-> Varias Columnas de la tabla

• Atributos multievaluado -> Nueva tabla dos columnas (C-1º de la entidad y el valor del atributo) ¿Cuál es la C-1º de la nueva tabla?

– Entidad Débil -> tabla + columna con la C-1º de la entidad fuerte de la que depende.

– Relación binarias -> tabla con los posibles atributos de la relación

• 1:1 -> en una de las dos entidades introduces la C-1º (clave externa) de la otra (la elección minimizará el número de NULLS). Otra opción si la participación es total en los dos sentidos se realiza una tabla mezclada con las dos entidades participantes (aunque esta última opción no se utilizará).

• 1:n -> Igual que 1:1 pero identificando la entidad que participa como n para añadir la C-1º de la otra entidad (clave externa).

• n:m -> se crea una tabla con las C-1ºs (claves externas) de las entidades de la relación y con los posibles atributos de la relación (reificación). ¿Cuál es la C-1º de la nueva tabla?

– Relaciones n-arias (n>2) -> tabla con una columna para cada atributo de las C1ºs (claves externas) de las entidades + posibles atributos de la relación.

1. Mapeado de los tipos de entidades regulares (EMPLEADO, DEPARTAMENTO, PROYECTO)

2. Mapeado de los tipos de entidades débiles (SUBORDINADO)

3. Mapeado de los tipos de relación 1:1 (ADMINISTRA)

– Metodología clave externa (minimizar NULLs) (siempre utilizaremos esta)

– Metodología de relación mezclada (participación parcial en los sentidos y se pueden mezclar las dos entidades en una misma tabla)

– Metodología de referencia cruzada (la misma que en m:n)

4. Mapeado de los tipos de relación 1:n (CONTROL, TRABAJA\_PARA, CONTROLA) (minimizar NULLs y redundancia) ´

5. Mapeado de los tipos de relación m:n (TRABAJA\_EN)

6. Mapeado de los atributos multivalor (LOCALIZACIONES\_DPTO)

7. Mapeado de los tipos de la relación n-aria

1. **Dependencias funcionales y normalización**

Una **superclave** X de un esquema de la relación R={A1 ,…,An } es un conjunto de atributos S ⊆ R con la propiedad de que no habrá un par de tuplas t1 y t2 en ningún estado de la relación permitido r de R tal que t1 [X] = t2 [X].

Un **clave** K es una superclave con la propiedad adicional de que la eliminación de cualquier atributo de K provocará que K deje de ser una superclave. Es la superclave mínima.

Si un esquema de una relación tiene más de una clave cada una de ellas se llama **clave candidata.**

Una de ellas se elige arbitrariamente como **clave principal.**

El resto de las claves candidatas son **claves secundarias.**

Cualquier atributo del esquema de la relación R que pertenece a una clave candidata o es una clave candidata se denomina **atributo primo** o primario.(que forma parte de una clave, pero no es la clave entera)

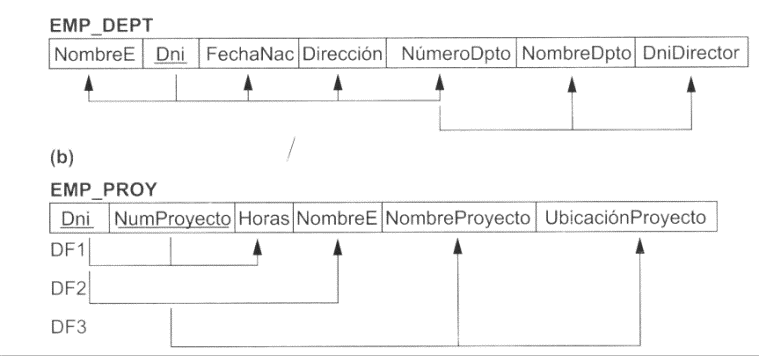
**Un atributo no** primo no es miembro de una clave candidata o no es una clave candidata.

Una **dependencia funcional (**DF) es una restricción implícita.

Si X e Y son dos conjuntos de atributos, decimos que hay un DF de X a Y o que Y depende funcionalmente de X, si y solo si siempre que dos tuplas que coinciden en su valor X, necesariamente deben coincidir en su valor Y.

Es decir, los atributos de Y están unívocamente determinados por los de X .

Notación: X → Y (dni → FechaNac, pero no FechaNac → dni).



Texto, Carta

Descripción generada automáticamente

Reglas de inferencia:

RI1 especifica que un conjunto de atributos siempre se determina a si mismo o cualquiera de sus subconjuntos (obvio). Ya que RI1 genera dependencias que siempre son verdaderas estas se llaman triviales, las no triviales son el resto.

RI2 o regla de aumento dice que añadir el mismo conjunto de atributos a cada lado de la dependencia genera otra dependencia válida.

RI3 o regla transitiva dice que una dependencia funcional X → Z en un esquema de relación R es una dependencia transitiva si existe un conjunto de atributos Y que ni es clave candidata ni es un subconjunto de una clave de R, y se cumple tanto X → Y como Y → Z.

RI4 (regla de eliminación de atributos del lado derecho), si aplicamos esta regla repetidamente podemos descomponer la DF {X → A1 , …, An } en el conjunto de DFs {X → A1 , …, X → An }. RI5(regla de unión de atributos), nos permite realizar lo contrario para combinar DFs {X → A1 , …, X → An } en la DF {X → A1 , …, An }.

Mínimos:

Dos conjuntos de dependencias F y E son equivalentes si toda dependencia de uno se puede inferir de las dependencias del otro y viceversa (es decir F + = E+ ).

Un conjunto de DFs de F es mínimo si:

– La parte derecha de todas sus dependencias es un solo atributo.

– Si eliminamos una dependencia, obtenemos un conjunto no equivalente a F.

– Si eliminamos un atributo en la parte izquierda de una dependencia, obtenemos un conjunto no equivalente a F. En otras palabras: DFs en forma canónica y sin redundancias.

Una cobertura mínima de un conjunto de DFs de F es un conjunto mínimo equivalente a F.

Texto, Carta

Descripción generada automáticamente

Imagen que contiene Logotipo

Descripción generada automáticamente

Ejemplo:

1. Establecer F := E

**/\* Descomponer en dependencias sobre atributos individuales en la parte derecha \*/**

2. Ya lo están.

**/\* Eliminar atributos que sobren en las partes izquierdas (atributos redundantes) \*/**

3. De la única que podemos quitar es de AB → D ¿Podemos quitar A o B de la parte izquierda quedando B → D o A → D? Es decir ¿podríamos sustituir AB → D por B → D o A → D? Supongamos la primera DF1: B → A, aumentando con B (RI2) inferimos BB → AB o lo que es lo mismo B → AB. Así tenemos estas dos DFs: {B → AB, AB → D} |= B → D (RI3). Por lo tanto si sustituimos AB → D por B → D tenemos lo mismo, ya que B → D se infiere de las dos DFs (es decir se infiere de E).

**/\* Eliminar dependencias que se infieren de otras (inferencia redundantes) \*/**

4. Hacer notar que tenemos {B → A, D → A, B → D}, así que {B → D, D → A} |= B → A y esta ya está, y por lo tanto es redundante. Así F:={B → D, D → A}

**/\* Devolver la cobertura mínima \*/**

1. return F (es una cobertura mínima)

Formas normales:

Forma normal 1:

Decimos que un esquema está **1NF** si:

– Los atributos son atómicos y univaluados.

– Los nombres de atributo son únicos.

– No hay tuplas duplicadas (consecuencia: todo esquema tiene alguna clave).

– El orden de tuplas y atributos es arbitrario .

Realmente estas propiedades se considera parte inherente del modelo relacional.

Aunque tenemos que hacer notar que:

– SQL sólo cumple la primera de estas condiciones (el tratamiento de NULL se sale también del modelo relacional)

– Se estudian alternativas como el modelo relacional anidado, que admite relaciones como valores de atributos

Esta Forma Normal elimina los valores repetidos en una BD

(**Si en la dependencia a→b, a es un atributo primo y b no es nada, entonces en 1NF**)

Forma normal 2:

Está basado en el concepto de DF total (si se elimina cualquier atributo de X se rompe la dependencia).

Una dependencia funcional X → Y es plena si no le sobra ningún atributo a X, es decir X – {A} no determina funcionalmente a Y, ∀A ∈ X.

Dicho de otro modo, los atributos dependen de la clave completa; sólo los atributos de una clave pueden depender de partes de éstas.

Atributo no primo o no primario = atributo que no es parte de ninguna clave.

Un esquema R está en 2FN si todo atributo no primo forma una DF total de alguna clave candidata.

Un esquema R es 2NF si todo atributo no primario de R tiene una dependencia funcional plena con las claves de R.

Recordar que atributos primarios son los que forman parte de alguna clave.

Dicho de otro modo un esquema R es 2NF: si cualquier atributo que no sea constituyente clave primaria, depende al menos de una clave candidata en vez de solo una parte de ella.

(**Si en la dependencia a→b, a no es nada y b no es nada, entonces en 2NF**)

Forma normal 3:

Se basa en el concepto de dependencia transitiva: una dependencia funcional X → Y en un esquema de relación R es una dependencia transitiva si existe un conjunto de atributos Z que ni es clave candidata ni es un subconjunto de una clave de R, y se cumple tanto X → Z como Z → Y.

Ejemplo: en la relación EMP\_DEPT la dependencia Dni → DniDirector es transitiva a través de NumeroDpto, ya que se cumplen las dependencias Dni → NumeroDpto y NumeroDpto → DniDirector (nota: NumeroDpto no es una clave por si misma ni un subconjunto de clave de EMP\_DEPT).

3FN = 2FN+ningún atributo no primo depende transitivamente de la clave principal o primaria a través de un atributo no primo.

(**Si en la dependencia a→b, b es atributo primo, entonces en 3NF**)

Forma normal Boyce-Codd:

Un esquema R es BCNF si para toda dependencia X → Y no trivial X es una superclave de R

Dicho de otro modo, no puede haber más que dependencia que con las superclaves.

Este esquema tiene por claves candidatas: IdPropiedad y {NombreMunicipio, NúmeroParcela}, se elige como primaria IdPropiedad.

(**Si en la dependencia a→b, a es una superclave, entonces en Boyce-Codd**)

Calculo y Algebra relacional:

Calculo:

Forma General: **{ t | F(t) }** conjunto de tuplas t tal que F(t) es verdadero.

Relación de intervalo (rango de la tabla A): **{ t | A(t)}** conjunto de tuplas que pertenecen a la relación A.

Proyección: **{ t.A1 , t.A2 | A(t) }** extraer los atributos de una tabla.

Selección: **{ t | A(t) and t.A1 >2000** } seccionar tuplas de una tabla.

Unión: **{ t | A(t) or B(t)}** (es necesario la compatibilidad de relaciones).

Intersección: **{ t | A(t) and B(t)}** (es necesario la compatibilidad de relaciones).

Resta: **{ t | A(t) and not B(t)}** (es necesario la compatibilidad de relaciones).

Producto cartesiano: **{ t, s | A(t) and B(s)}.**

Join: **{ t, s | A(t) and B(s) and t.A1<s.B1}**

Join: **{ t | A(t) and ((∃s)(B(s) and t.A1 s.B1}**

Equi-Join: **{ t, s | A(t) and B(s) and t.A1=s.B1 }**

Equi-Join**: { t | A(t) and ((∃s)(B(s) and t.A1=s.B1 ))}**

**Ejemplos de consultas:**

Liste el nombre y la dirección de todos los empleados que trabajan para el departamento ‘Investigación’:

{t.Nombre, t. Apellido1, t.Dirección | EMPLEADO(t) AND (∃d) (DEPARTAMENTO(d) AND d.NombreDpto=‘Investigación’ AND d.NumeroDpto=t.Dno) }

Por cada proyecto ubicado en ‘Gijón’, obtenga el número de departamento que lo gestiona, los apellidos, la fecha de nacimiento y la dirección del director del mismo:

{p.NumProyecto, p. NumDptoProyecto, e.Apellido1, e.FechaNac, e.Dirección | PROYECTO(p) AND EMPLEADO(e) AND p.UbicacionProyecto=‘Gijón’ AND ((∃d)(DEPARTAMENTO(d) AND p.NumDptoProyecto=d.NumeroDpto AND d.DniDirector=e.Dni))}

Algebra:

Toda operación de selección cumple:

– Los atributos de la selección son los mismos que los de la relación.

– El número de tuplas resultante es siempre menor o igual que el número de tuplas de la relación.

– La operación selección es conmutativa:

• σcondición1 (σcondición2 (R)) = σconcición2 (σcondición1 (R)) = σcondición1 and condición2 (R)

• Esto se puede aplicar a una cascada de condiciones

En general la operación selección es una elección de filas de las relaciones (partición horizontal), al contrario de la operación que vamos a estudiar a continuación de PROYECCIÓN. Está segunda operación unaria selecciona ciertas columnas de una tabla.

Es por que la operación PROYECCIÓN se suele ver como una partición vertical de la tabla sobre la que opera.

ω En general: π<lista de atributos> (R)

Operaciones entre conjuntos:

UNIÓN: R ∪ S es una relación que incluye todas la tuplas que están en R o están en S o en ambas R y S (obviamente no hay duplicados, “o”).

INTERSECCIÓN: R ∩ S es una relación que incluye todas la tuplas que están en R y S . DIFERENCIA: R-S es una relación que incluye todas la tuplas que están en R pero no están en S.

Producto Cartesiano: Se trata también de una operación de conjuntos binarios, aunque los operandos no tienen porque ser de unión compatible.

Ejemplo de la operación UNIÓN: Recuperar los Documentos Nacionales de Identidad de todos los empleados que, o bien trabajan en el departamento 5 o bien supervisan a estos:

Diagrama

Descripción generada automáticamente1. DEP5\_EMPS← σDno=5 (EMPLEADO)

2. RESULTADO1 ← πDni (DEP5\_EMPS)

3. RESULTADO2(Dni) ← πSuperDni (DEP5\_EMPS)

4. RESULTADO ←RESULTADO1 ∪ RESULTADO2

σc (R) ∩ σd (S) = σc and d (R ∩ S)

σc (R) ∪ σd (R) = σc or d (R)

Producto cartesiano:

Producto Cartesiano: Se trata también de una operación de conjuntos binarios, aunque los operandos no tienen porque ser de unión compatible.

En general R(A1 , A2 ,…, An ) × S(B1 , B2 ,…, Bm) es una relación Q de grado n+m atributos Q(A1 , A2 ,…, An ,B1 , B2 ,…, Bm). ω En la relación Q resultante tiene una tupla por cada combinación de éstas (una para R y otra para S).

Por tanto si R tiene nR tuplas y S tiene nS tuplas entonces R × S tendrá nR \* nS tuplas. Supongamos la siguiente secuencia de operaciones:

1. EMPLEADAS\_FEMENINAS ← σSexo=‘M’ (EMPLEADO)

2. NOMBRE\_EMPLEADOS ← πNombre, Apellido1, Dni(EMPLEADAS\_FEMENINAS)

3. EMPLEADOS\_SUBORDINADOS ← NOMBRE\_EMPLEADOS × SUBORDINADO

4. SUBORDINADOS\_ACTUALES ← σDni=DniEmpleado (EMPLEADOS\_SUBORDINADOS)

5. RESULTADO ← πNombre, Apellido1, NombreSubordinado(SUBORDINADOS\_ACTUALES)

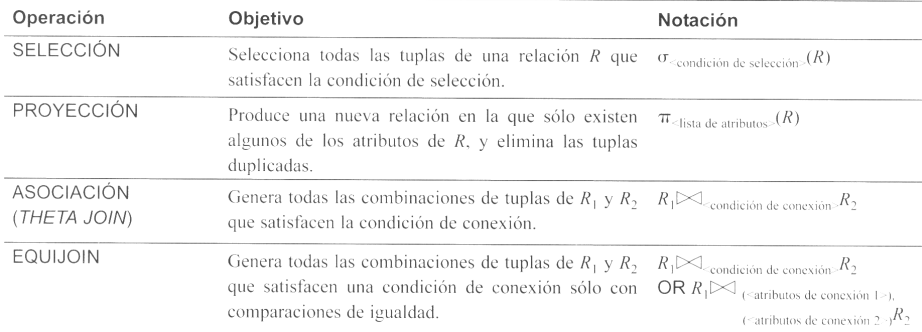
El producto cartesiano solo tiene sentido si se hace a continuación que combine las relaciones componentes en la manera que deseamos.

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente



Tabla

Descripción generada automáticamente

Operaciones :

– SUMA (SUM)

– MEDIA (AVERAGE)

– MÁXIMO (MAXIMUM)

– MÍNIMO (MINIMUN)

– CONTAR (COUNT)

Texto

Descripción generada automáticamente

Ejemplos consultas:

El número de empleados del mismo, y la media de sueldos, renombrando los atributos resultantes:

ρR(Dno, NumEmpleados, MediaSueldos) (Dnoℑ COUNT Dni, AVERAGE Sueldo (EMPLEADO)).

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

1. **Implementación**

Las tablas de las BDs se transforman de campos y registros:

Tupla → registro

atributo → campo

Delimitación de campos y registros:

Longitud fija:

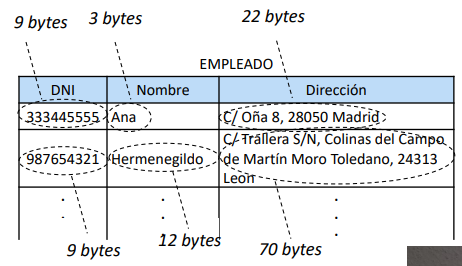
Longitud variable:

separador

indicador de longitud.

fichero de direcciones.

Cabeceras con autodescripción de estructuras, nº de registros



Longitud fija:

Diagrama, Tabla

Descripción generada automáticamente

Longitud variable:

Tabla

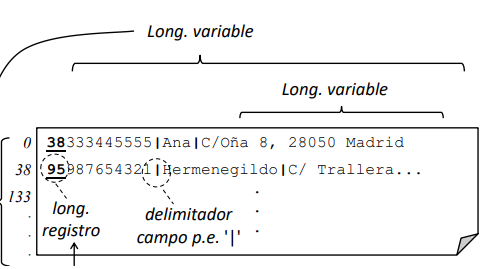
Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama, Texto

Descripción generada automáticamente



Longitud fija:

Acceso a datos más rápido:

Los registros se direccionan por número relativo de registro (nrr).

Longitud variable:

Acceso más lento:

Pero: Optimización de espacio en disco.

Los registros se direccionan por su posición en bytes (offset).

Cabecera (necesidad para la auto-descripción, metadatos).

Sección descriptiva de la estructura física de un archivo: qué campos, cómo se delimita, etc.

Esta estructura de metadatos permite implementaciones más generales que admitan diferentes formas de organizar físicamente los registros

En general:

– LF: Rapidez de acceso y desaprovechamiento de espacio.

– LV: Lentitud de acceso y aprovechamiento de espacio.

Longitud fija: economía de procesamiento.

Solución cómoda de desarrollar y ejecutar.

Longitud variable: economía de espacio en disco.

El esfuerzo de desarrollo / ejecución puede compensar, por ejemplo, cuando se necesita ahorrar espacio.

El consumo de disco es un problema

La variabilidad de tamaño de registros es muy amplia: campos muy variables y/u opcionales (valores NULL)

Se juntan registros de diferentes tipos en el mismo archivo

Si un campo es realmente grande (documento de texto o multimedia) se suele almacenar en un archivo aparte y se guarda en el registro un descriptor de la dirección donde se almacena el valor del campo.

El HW determinan el coste de las operaciones de acceso a los datos.

Existen diferentes tipos dispositivos de almacenamiento de datos:

Memoria principal (RAM)

Máxima velocidad de acceso: ns.

Alto coste económico comparativo con otros dispositivos y por tanto limita su disponibilidad.

No persistente.

Disco magnético (es el que normalmente se utiliza)

Sigue siendo el dispositivo de almacenamiento más común para bases de datos.

Órdenes de magnitud más lento que la memoria RAM, básicamente debido a la necesidad de movimiento mecánico, del orden de ms.

Basaremos nuestro análisis de costes en este tipo de soporte.

Memoria flash / SSD

“Solid state” alude al contraste con piezas móviles.

Lectura casi tan rápida como la memoria RAM.

Escritura más lenta, y requiere borrado previo.

Nº limitado de reescrituras (p.e. varios miles).

Costes de acceso:

Operaciones elementales de lectura/escritura y su coste

Seek: movimiento radial (caso promedio: desplazamiento de 1/3 del radio)

Latencia rotacional (caso promedio: 1/2 rotación)

Transferencia (nº de bloques a leer / nº de bloques por pista × tiempo rotación)

Lectura (o escritura) de un archivo

Posicionarse en el primer sector: seek + latencia

Recorrer los bytes: tiempo de transferencia de los bloques que ocupa el archivo

Saltos adicionales por fragmentación: seek + latencia por cada bloque no contiguo que contenga el archivo

Estrategias de optimización: Evitar leer datos innecesarios, evitar la repetición de lecturas, reducir el nº de seek + latencia, por ejemplo utilizando buffers, y muchas otras estrategias en el diseño de algoritmia

Ejemplo de diferencias entre lectura secuencial de registros vs acceso aleatorio:

– 1) Supongamos que todos los clusters están llenos (lectura secuencial de registros), o – 2) que cada cluster solo contiene un registro (lectura aleatoria de registros).

Supongamos que tenemos un HD de 40 sectores por pista, con 512 bytes por sector, con seek promedio de 9,5 ms, con 8 sectores por cluster y con una velocidad de rotación de 5400 rpm. Supongamos un fichero para leer de 2M con 8000 registros de 256 bytes.

Así en el primer caso la lectura de cada cluster involucra los siguientes tiempos:

– Posicionamiento en la pista: seek promedio de 9,5ms

– Latencia rotacional: tiempo en dar una vuelta 60´/5400=11,1 ms, así en promedio será 11,11/2=5,5 ms de latencia rotacional.

– Tiempo de lectura de un cluster: (8/40)x11,1=2,2 ms

– Así el tiempo total para la lectura de un cluster es: 17,2 ms

Cuantos clusters tengo que leer: como un cluster contiene 8 sectores de 512 bytes, entonces un cluster son 4096 bytes. Así el fichero de 2M, es decir de 2048000 bytes, cabe en 2048000/4096=500 clusters.

Por tanto el tiempo en leer los 500 clusters son: 17,2 ms x 500 = 8600 ms = 8,6 s ω Ahora en el segundo caso suponiendo que por cada cluster contiene un único registro, entonces tenemos que leer un total de 8000 registros y por tanto el tiempo será:

– Posicionamiento en la pista: seek promedio de 9,5ms

– Latencia rotacional: tiempo en dar una vuelta 60´/5400=11,1 ms, así en promedio será 11,1/2=5,5 ms de latencia rotacional.

– Tiempo de lectura de un cluster: (8/40)x11,1=2,2 ms

– Así el tiempo total para la lectura de un cluster es: 17,2 ms

Pero ahora hay que leer 8000 clusters (el fichero tiene 8000 registros de 256 bytes), ya que no están llenos y su tiempo total de lectura será: 17,2 ms x 8000 = 137600 ms = 137,6 s

La diferencia es notable para la lectura del mismo fichero cuando los clusters no están llenos (fagmentación interna de clusters): 8,6 s vs 137,6 s.

Supongamos que queremos leer un fichero de 9K, es decir compuesto por 36 registros de 256 bytes.

Y supongamos que los registros están llenando los clusters a lo máximo que puedan estar llenos.

El tiempo para la lectura del fichero será:

– Posicionamiento en la pista: seek promedio de 9,5ms

– Latencia rotacional: tiempo en dar una vuelta 60´/5400=11,1 ms, así en promedio será 11,1/2=5,5 ms de latencia rotacional.

– Tiempo de lectura de un cluster: (8/40)x11,1=2,2 ms

– Así el tiempo total para la lectura de un cluster es: 17,2 ms

Ahora cuantos clusters ocupa el fichero: como un cluster son 4K el fichero ocupa 3 clusters, 4k en un cluster, 4k en otro y 1k en otro.

Pero hay que leer los tres clusters ya el cluster e la unidad mínima de lectura/escritura. Por tanto el tiempo de lectura es: 17,2 ms x 3 = 51,6 ms.

El acceso a los datos puede ser de dos formas:

– Posición: se identifican los registros por el número de su posición en bytes. El acceso es directo cuando se utiliza la posición: se conoce el número de bytes que hay que saltar hasta llegar a un determinado registro. Así el acceso al registro es de O(1) cuando se conoce exactamente en que posición de byte está. Las dos formas de conocer esa posición:

• NNR: para registros de LF.

• OFFSET: Para registros de LV.

– Clave: En vez de identificar registros por el nº de su posición se usa una clave.

• La clave puede ser uno de los campos, combinación de campos o derivados de los campos y normalmente se expresan en forma canónica.

• Pueden existir varios tipos de claves para identificar registros:

– Clave primaria: identifican el registro de manera única.

– Clave secundaria: no tienen porque identificar los registros de manera única. Son campos que pueden tener valores iguales en otros registros.

Añadir nuevos registros: Suponiendo que los registros no tienen un orden particular en el disco, los nuevos registros se añaden al final de la tabla según van llegando.

Aunque cuando hay huecos, esto cambia.

Eliminar registros: cuando eliminamos registros el objetivo es volver a usar el espacio que ocupaba el registro, y se procede así:

– Al eliminar un registro se marca como libre el espacio que ocupaba, con una marca al principio del registro:

• Se puede utilizar el primer campo.

• Se puede utilizar un campo especial reservado (así se puede recuperar el registro: undelete).

– Así los registros borrados se convierten en huecos para ser utilizados cuando se añaden nuevos registros.

– Generalmente se produce compactación de huecos cada cierto tiempo.

Diagrama

Descripción generada automáticamente

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

Ejercicio ej:

Tenemos un fichero con **106 de registros** **de longitud fija de 1.000 bytes** cada uno en un disco magnético con las siguientes características:

• **100 sectores por pista.**

**• 8 sectores por bloque.**

**• 500 bytes por sector.**

**• Un seek promedio de 9 ms.**

**• Una velocidad de giro de 10.000 rpm.**

Estimar el tiempo necesario para leer el fichero completo en los supuestos que se describen a continuación. Respecto a la fragmentación del fichero:

**a. Mejor caso: los datos del fichero ocupan una región compacta del disco (es decir, un** supuesto ideal en el que la fragmentación es nula).

**b. Peor caso: el fichero está fragmentado al máximo**

Respecto al programa que lee los datos:

**i. Peor caso: los registros se leen de uno en uno.**

**ii. Mejor caso: se lee el fichero completo en una sola instrucción.**

**iii. Término medio: se leen los datos utilizando un buffer de lectura de 80.000 bytes.**

Los cálculos se pueden organizar por registro, o mejor, por fragmentos que se van a leer sin “saltos” entre medias, y finalmente multiplicar el coste por fragmento por el número de fragmentos. La cuestión es por tanto identificar en cada caso qué fragmentos son ésos. También es posible separar el cálculo de coste por operaciones seek + latencia por un lado, y transferencia por otro, ya que el coste por transferencia es independiente de los saltos que se realicen entre lecturas. Aquí vamos a seguir esta opción.

Hacemos algunos cálculos preliminares útiles para todos los casos:

* **La velocidad de rotación en revoluciones por milisegundo es 10.000rpm / 60 / 1000 = 1/6 rotaciones por ms (una rotación completa tarda 6ms y una promedio 3ms).**
* **Cada operación de seek + latencia supone 9ms + 3ms = 12 ms.**
* **Cada bloque tiene 8 · 500 bytes = 4.000 bytes.**
* **En cada bloque caben pues 4.000 bytes / 1.000 bytes = 4 registros.**
* **El fichero ocupa 106 registros / 4 registros por bloque = 250.000 bloques.**
* **en el buffer del caso iii. caben 80.000 bytes / 1.000 registros = 80 registros; caben los datos de 80.000 bytes / 4.000 bytes = 20 bloques.**

a. Al no haber fragmentación, las únicas operaciones de seek + latencia se deberán a “interrupciones” en la lectura producidas en el programa.

i. Cada registro da lugar a un seek + latencia. Se lee el bloque completo en el que esté almacenado el registro (a pesar de que en un bloque caben 4 registros), pues es la mínima unidad de lectura/escritura en cualquier caso, lo que da lugar a una parte de lectura repetida. Se producen por tanto un millón de lecturas de bloques (una por registro). De aquí se derivan pues: ´

**• Seek + latencia: 106 · 12ms = 12.000s = 200 min**

**• Transferencia: 106 \* tiempo de 1 rotación \* número bytes por transferir / número bytes por pista = 106 \* 6ms\*4.000 bytes/(100 sectores por pista \* 500 bytes por sectores) = 8min**

**• Tranferencia: 106 \*tiempo de rotación\*número de bloques a leer / número bloques por pista) = 106 \* 6ms \* 1bloque / (100/8 bloques por pista) = 8 min Total: 208 min**

ii. Se produce un único (y necesario) movimiento de seek + latencia (9ms) y el programa no produce ninguno adicional. El coste de transferencia es el estrictamente necesario para leer los datos (una sola vez, sin redundancia): ´

**• Seek + latencia: 12ms**

**• Transferencia: 106registros · 1.000bytes / 50.000bytes (por pista) · 6ms=2min**

**• Transferencia: 6\*250000/(100/8)= 2min**

**Total: 2 min 12 ms 1**

iii. El buffer se llena sin interrupciones por parte del programa. Así pues se producirá únicamente un seek + latencia cada vez que se llene el buffer. Para leer todos los registros, el buffer se tiene que leer **(106 registros / 80 registros por buffer) = 12.500** veces. La transferencia se produce sin repetición de lecturas, pues el tamaño del buffer excede el tamaño de bloque, y es un múltiplo exacto de éste. Tenemos pues:

**• Seek + latencia: 12.500 · 12ms = 2 min 30 s**

**• Transferencia: 2 min (igual que para el caso ii)**

**Total: 4 min 30 s**

b. Como mínimo se va a producir un movimiento seek + latencia por bloque.

i. Como cada registro cabe en un bloque, el hecho de que los bloques estén separados resulta indiferente. El coste es el mismo: 208 min. ´

ii. El coste de transferencia no cambia respecto al caso **a.**, pero se producen seek + latencia adicionales: uno por bloque.

**• Seek + latencia: 12ms · 250.000 bloques = 50 min**

**• Transferencia: 106 registros · 1.000 bytes / 50.000 bytes (por pista) · 6 ms = 2 min Total: 52 min**

iii. El buffer no evita los seek + latencia por los cambios de bloque que tienen lugar dentro del llenado del buffer. Se evitan las lecturas redundantes del caso i, y por lo demás todo resulta igual que en el caso ii: **52 min**

Reutilización dinámica de registros de longitud fija:

– Cuando se añade un nuevo registro se sitúa rellenando los posibles huecos que existan.

– Sino hay huecos al final.

– Por lo tanto hay que buscar huecos:

• Recorridos secuencial es muy lento.

• Pila de registros vacíos (implementación de la pila mediante una lista enlazada de NNRs vacios):

– Se utiliza la cabecera para guardar el primer registro libre: NNR0.

– Se utilizan los campos libres para guardar el siguiente NNR borrado.

Reutilización dinámica de registros de longitud variable:

– El método es el mismo que con registros de longitud fija, pero se utiliza el OFFSET en lugar del NNR.

– Conviene tener un contador de longitud de registro al principio de cada registro.

– Ahora hay que recorrer la lista de huecos, y encontrar el hueco apropiado.

– Por tanto el contador se ajusta con el tamaño del nuevo registro.

– El espacio que sobra se puede convertir en un nuevo hueco (contador, marca y siguiente hueco), agregándose a la lista de huecos.

– Así se evita fragmentación interna:

Puede surgir dos tipos de fragmentación:

– Interna: Espacio desaprovechado dentro de los registros.

• con LV no hay a penas, o al menos se intenta evitar (ejemplo anterior).

– Externa: Espacio desaprovechado por los huecos dejados por los registros.

• Con LF es igual a los registros borrados aún no reutilizados.

Interesa frenar al máximo esa fragmentación, así se existen diferentes estrategias para seleccionar los huecos en la lista en registros de LV:

**– FIRST FIT: Lista de huecos sin ordenar(ultimo borrado)**

**– BEST FIT: Lista de huecos ordenadas de más pequeños a más grandes.(hueco mas pequeño)**

**– WORST FIT: Lista de huecos ordenadas de más grandes a más pequeños.(hueco más grande)**

En último término la fragmentación se elimina mediante compactación.

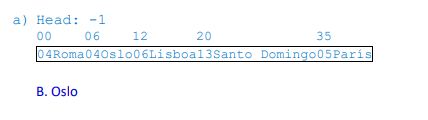
Ejemplo ejercicio:

Supongamos que borramos Oslo, insertamos Atenas, borramos Santo Domingo, borramos Lisboa, e insertamos Berlín, en este orden. Mostrar el estado del fichero, con la lista de espacios libres, antes y después de la inserción de Berlín, según las estrategias de

a) first-fit y

b) worst-fit.

Indicación: para simplificar, supóngase que tanto el indicador de longitud como los punteros de la lista de espacios libres ocupan 2 bytes.



Escala de tiempo

Descripción generada automáticamente

Escala de tiempo

Descripción generada automáticamente

Técnicas principales de indexación para BDs:

Índices simples: el mapping lo da un array –en RAM– de pares:

valor campo / dirección (o lo que es lo mismo pares de CLAVE/LOCALIZACIÓN).

Árboles B: semejantes a los índices simples, pero en vez de un array, la estructura es más sofisticada (un árbol – en disco – con técnicas de actualización elaboradas):

Es una forma automática de hacer un caché o búfer en RAM, de un índice en forma de árbol en disco duro:

Árboles B, B virtuales, B\*, B+, etc.

Índices:

Índice primario:

Operaciones Ram

Operaciones disco

Crear índice

– Leer fichero de datos (secuencial por bloques) y construir array en RAM

– Ordenar array

– Guardar (secuencial por bloques) array en archivo

Cargar índice en RAM (si es posible)

– Lectura (secuencial por bloques) del archivo de índice → índice en RAM (crear array en RAM que contenga el fichero indexado).

Búsqueda de un registro

– Búsqueda (binaria) en índice → dirección (nnr o offset) del registro buscado

– Acceso (seek + read) al registro sabiendo su posición (localización física)

Añadir o Inserción de registro

– Archivo de datos: escribir el registro en la dirección d que corresponda (puede ser un hueco)

– Índice: añadir entrada manteniendo el orden del índice en la memoria

Eliminación de registro

– Archivo de datos: eliminar por el método de gestión del espacio que se desee

– Índice: eliminar entrada correspondiente en RAM

– De vez en cuanto es posible hacer Compactación → reconstruir el índice Modificación de un campo

– Archivo de datos: modificar el valor del campo

– Índice: si el campo es parte de la clave primaria: eliminar + insertar en el índice

– Se mueve el registro (si ya no cabe el registro) → actualizar su dirección en el índice (o eliminar + insertar)

– Otros datos: no implica reordenamiento en el índice

Índice secundario:

Búsqueda de registro

– Buscar clave secundaria en índice secundario → clave primaria

– Buscar en índice primario → dirección del registro en archivo de datos

– Acceder al registro en disco, leer

Inserción de registro

– Archivo de datos: insertar el registro

– Índice: análogo al índice primario

Eliminación de registro

– Archivo de datos: eliminar el registro

– Índice: se puede posponer la eliminación (técnica en diferido)

– El intento de acceso a un registro borrado se detecta a nivel del índice primario:

• El índice primario actúa como un filtro de seguridad entre índices secundarios y ficheros de datos, i.e. si en el I2º hay una C1º que no existe en el fichero, cuando acceda al I1º se dará cuenta que no existe y es cuando se elimina del I2º.

– Periódicamente, limpiar el índice secundario

– Compactación → el índice secundario no cambia

Modificación de un campo

– Archivo de datos: modificar

– Índice: si afecta a la clave secundaria, recolocar para mantener el orden

– Si afecta a la clave primaria, actualizar en el secundario la clave cambiada

– Si la clave primaria aparece en entradas de clave secundaria repetida, recolocar para mantener el orden de la clave primaria

– Si afecta a otros campos: el índice secundario no cambia, aunque se moviese el registro

Texto, Carta

Descripción generada automáticamente

**Arboles:**

Propiedades árbol b:

Propiedades de un árbol B:

1. Siendo k el número máximo de claves por página, el número máximo de descendientes por página es k+1.

2. Cada página excepto la raíz y las hojas tiene por lo menos ⌊k/2⌋+1 descendientes.

3. La raíz tiene por lo menos 2 descendientes a menos que sea una hoja.

4. Todas las hojas aparecen en el mismo nivel.

5. Una página que no es hoja con m descendientes contiene m-1 claves.

6. Una página hoja contiene por lo menos ⌊k/2⌋ claves y no mas de k.

Contenido de paginas:

Contenido de una página

– Claves ordenadas acompañadas por dirección de página hija; cada clave “separa” dos páginas hijas

– Las claves de cada página son “mayores” que las de la página que la “precede”

– La clave que separa dos páginas es mayor que las claves de una y menor que las claves de la otra

– Las direcciones de página en las páginas hoja son “null”

Diagrama

Descripción generada automáticamente

Árbol B costes:

Texto

Descripción generada automáticamente

El coste de la búsqueda y de la inserción es 𝑂(ℎ), con ℎ = altura del árbol

La altura del árbol es mayor cuanto menos llenas estén las páginas.

Si 𝑘 es el máximo nº de claves por página, 𝑚 es un mínimo que aseguremos, y 𝑛 es el nº total de claves en el índice, se puede ver que la altura del árbol es…

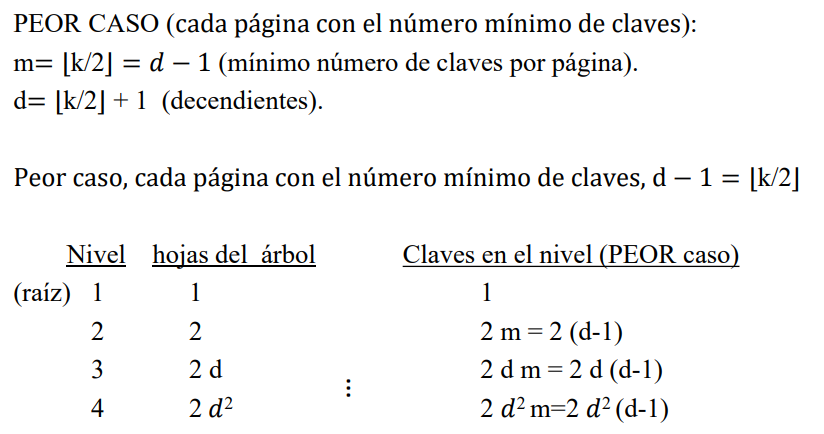
ℎ𝑚𝑖𝑛 ≤ ℎ ≤ ℎ𝑚𝑎𝑥

log𝑘+1 𝑁 + 1 ≤ ℎ ≤ 1 + log𝑚+1 (𝑁 + 1)/2)

Por tanto, a mayor 𝑚, menor coste en las operaciones en disco duro.

Con la técnica básica de inserción que hemos visto, aseguramos un mínimo de por página.

Por ejemplo con k=512 y N=1.000.000 implica ℎ ≤ 3.37 (muy pocos accesos a disco duro)



Tabla

Descripción generada automáticamente

Añadir:

Ya sabemos de los ejemplos de antes, el procedimiento que hemos realizado lo podemos resumir por:

I. Buscar la página hoja donde debería insertarse la clave

• Es necesario controlar las entradas duplicadas si el árbol es un índice primario, si es uno secundario no.

II. Si hay espacio en la página para una clave más, insertar, se termina la inserción.

III. Si no lo hay, dividir la página en dos, dejando una clave + nº página entre medias.

IV. Si la página era la raíz, crear una nueva raíz (aumenta la altura del árbol), se termina la inserción. Si no, subir la clave intermedia + nº de página a la página “padre”, volver al paso II.

Ejemplo:

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Eliminación:

1. Buscar clave.

2. Si no está en una hoja, intercambiar con la hoja adecuada.

3. Eliminar la clave.

4. Si la página no está bajo mínimo, fin.

5. Si está bajo mínimo:

a) Redistribuir si hay una página hermana (inmediatamente contigua) con una clave de sobra.

b) Concatenar en otro caso (con una página “hermana” inmediatamente contigua). La concatenación puede propagarse hacia arriba (si se propaga hasta la raíz, se reduce en un nivel la altura del árbol).

Diagrama

Descripción generada automáticamenteEjemplo:

Ejemplo de inserción:

Diagrama

Descripción generada automáticamente

Árbol B\*:

División 2 → 3:

• Hacer inserción con redistribución.

• Cuando se llenan dos páginas contiguas, partirlas en tres:

• Así las páginas ⌊2k/3⌋ llenas en lugar de ⌊½k⌋

• Así, cada página excepto la raíz y las hojas tienen al menos ⌊2k/3⌋ descendientes.

• Asegurar este mínimo también al eliminar (concatenar “tres a dos”).

• Raíz (caso especial): no tiene hermanas:

• La raíz tiene al menos 2 descendientes (a menos que sea una hoja).

• Página raíz puede ser más grande para mantener la proporción (1/2 raíz = 2/3 página estándar).

• O bien división 1 → 2 ⇒ páginas bajo raíz pueden estar sólo a la mitad.

• O bien, se permite (necesariamente) tener menos claves a la raíz (mínimo 1) y sus hijas (mínimo ⌊𝑘∕2⌋)

Diagrama

Descripción generada automáticamente

Ejemplos:

Inserción sin distribución:

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamenteCon redistribución:

Imagen que contiene reloj

Descripción generada automáticamente

Ejemplos:

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Árbol B+:

Diagrama

Descripción generada automáticamente

Ejemplos:

Diagrama, Dibujo de ingeniería

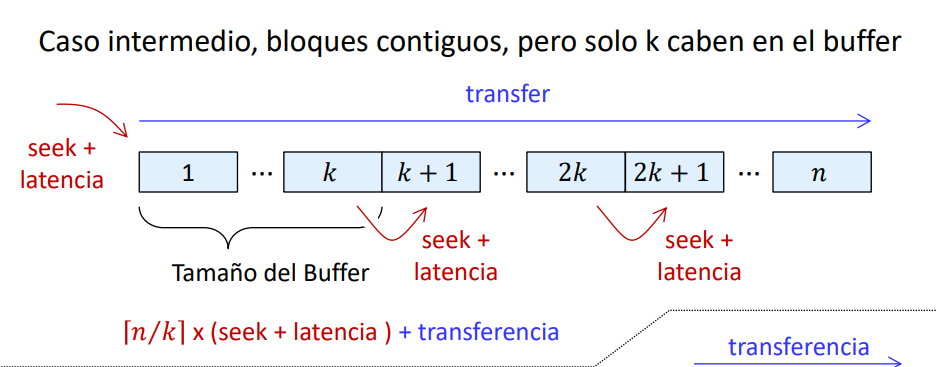
Descripción generada automáticamente

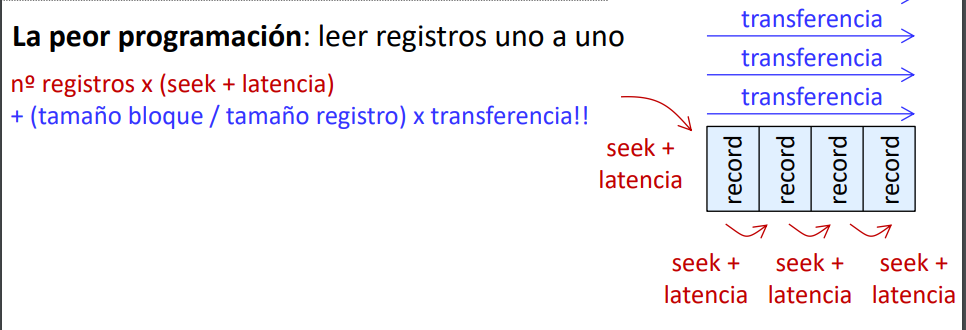
Tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente





Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Diagrama, Tabla

Descripción generada automáticamente

Diagrama, Tabla

Descripción generada automáticamente

Diagrama, Tabla

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

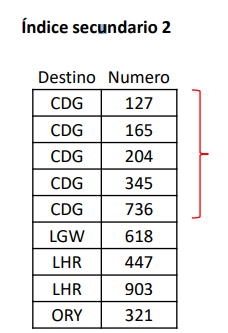
Diagrama

Descripción generada automáticamente

Tabla

Descripción generada automáticamenteTabla

Descripción generada automáticamente



Diferencias entre índices primarios y secundarios ω Índices siempre ordenados por clave – En los secundarios cuando se repite la secundaria, ordenar por primaria ω Las direcciones físicas de registros en el índice primario son:

– Nrr cuando los registros se almacenan con longitud fija

Diagrama

Descripción generada automáticamente– Offset en bytes cuando se utiliza longitud variable ω Los índices simples tienen que caber en RAM

Diagrama

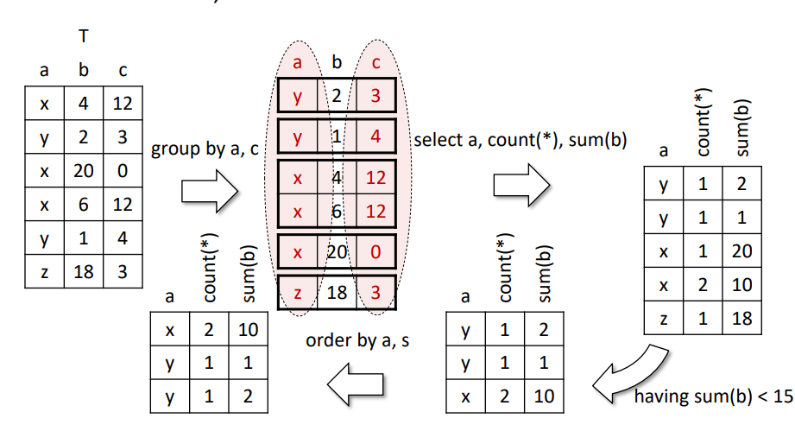
Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente



a. Mejor caso: los datos del fichero ocupan una región compacta del disco (es decir, un supuesto ideal en el que la fragmentación es nula).

b. Peor caso: el fichero está fragmentado al máximo Respecto al programa que lee los datos:

i. Peor caso: los registros se leen de uno en uno.

ii. Mejor caso: se lee el fichero completo en una sola instrucción.

iii. Término medio: se leen los datos utilizando un buffer de lectura de 80.000 bytes

**a.i. REGISTROS X (SEEK + LATENCIA + TRANSFERENCIA(BLOQUE) = TIEMPO (ms)**

**TRANSFERENCIA = (SECTORES POR BLOQUE) X (1/BLOQUES POR PISTA)**

**a.ii. SEEK + LATENCIA + TRANSFERENCIA = TIEMPO (ms)**

**TRANSFERENCIA = (SECTORES POR BLOQUE) X REGISTROS X BLOQUE POR REGISTRO / BLOQUE POR PISTA**

**a.iii. N2º + N3º = TIEMPO (ms)**

**LEER BUFFER = REGISTROS X BYTES(REGISTRO) / BYTES (BUFFER) = Nº**

**SEEK + LATENCIA X Nº = N2º**

**TRANSFERENCIA = REGISTROS X (BYTES(REGISTRO/BYTES(PISTA)) X TIEMPO ROTACION = N3º**

**b.i. IGUAL QUE EL a.i**

**b.ii. Nº + N1º = TIEMPO (ms)**

**SEEK + LATENCIA + REGISTROS + (BLOQUES POR REGISTRO) = Nº**

**TRANSFERENCIA = (SECTORES POR BLOQUE) X REGISTROS X BLOQUE POR REGISTRO / BLOQUE POR PISTA = N1º**

**b.iii. IGUAL QUE b.ii**