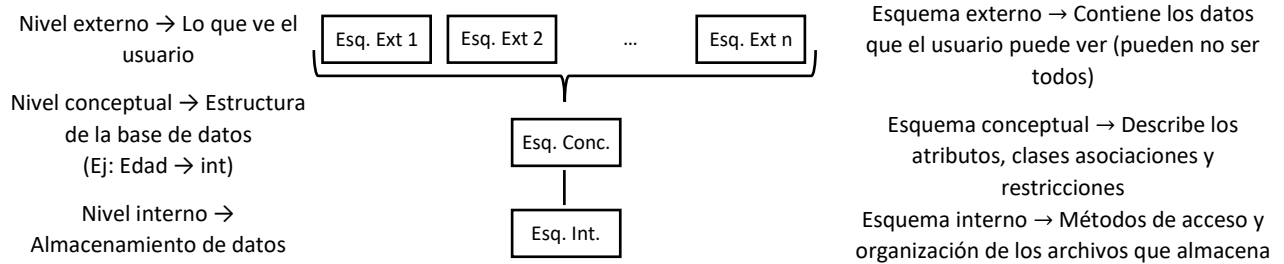


# Base de Datos

<https://github.com/AdriCri22/Base-Datos-BD-FIB>

## Tema 1 - Introducción



Independencia lógica:

- Cambios en el esquema conceptual no afectan a los esquemas externos que no hagan referencia a las clases, atributos o asociaciones modificadas (Ej: añadir un atributo no afecta a la vista).
- Cambio en un esquema externo no afectan a los otros esquemas, ni a los esquemas conceptual ni interno (Ej: añadir/eliminar vistas no afecta a las vistas independientes a esta).

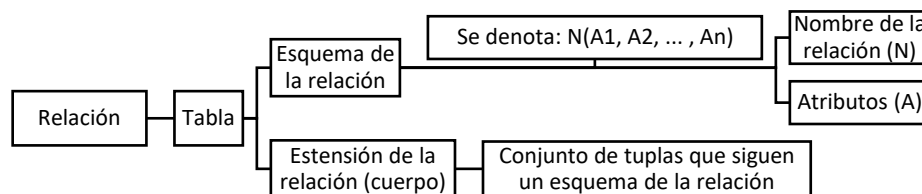
Independencia física:

- Cambios en el esquema interno no afecta a ninguno de los otros dos esquemas (Ej: cambio en el tamaño de las páginas, métodos de acceso).

## Tema 2 – Modelo Relacional

**Tupla** → Fila (Cardinalidad = nº tuplas)

**Atributo** → Columna (Grau = nº atributos)



**Dominio** → Conjunto de valores atómicos (Ej: int, string...)

En BD cuando un valor es NULL significa que el valor inexistente, es desconocido.

Nombre de la Relación				Atributos	
EMPLEADOS	DNI	Nombre	Teléfono		
	40.444.255	María Domínguez	NULL	←	→ Tupla
	33.567.711	Pere Roca	NULL		
	77.232.144	Elena Pla	934452435		

Dominio: Numeros\_de\_DNI  
 Dominio: char(n)  
 Dominio: integer

Conjunto atómico definido por el usuario

EMPLEADOS(dni, nombre, telefono)

### Reglas de integridad

- Regla de integridad de entidad: claves primarias únicas y sin valor NULL.
- Regla de integridad referencial: no puedes tener un valor en una foreign key (atributo de una tabla que referencia a otro de otra tabla) que no exista en la tabla.
  - **Restrict** → No permite borrar o modificar la clave primaria referenciada en alguna foreign key.
  - **Cascade** → Cuando se borra o modifica una clave primaria referenciada en alguna foreign key se borran o modifican todas las referencias.
  - **Anulación** → Al borrar o modificar una clave primaria referenciada en alguna foreign key, se ponen a NULL todas las referencias.
- Regla de integridad de dominio: respetar el tipo de valor (Ej: integer → introducir un int).

## Tema 3.1 – Lenguajes: SQL

Unión  $\rightarrow R = TABLA1 \cup TABLA2$

Se aplica a relaciones que sean compatibles (tienen esquemas con un conjunto de atributos idéntico y los dominios de cada pareja de atributos son los mismos). La unión resulta en una tabla con las tuplas de TABLA1 y TABLA2.

Renombramiento  $\rightarrow R = TABLA \{atributo1 \rightarrow atributoA, atributo2 \rightarrow atributoB\}$

La relación resultante no cambia solo cambian el nombre de los atributos.

Intersección  $\rightarrow R = TABLA1 \cap TABLA2$

Se aplica a relaciones compatibles y como resultado obtenemos las tuplas que tienen en común TABLA1 y TABLA2.

Diferencia  $\rightarrow R = TABLA1 - TABLA2$

Se aplica a relaciones compatibles y como resultado tenemos las tuplas de TABLA1 que no están en TABLA2.

Producto cartesiano  $\rightarrow R = TABLA1 \times TABLA2$

El resultado es una tabla que tiene los atributos de TABLA1 y TABLA2 combinando las tablas haciendo que haya tuplas de TABLA1  $\times$  tuplas de TABLA2.

Selección  $\rightarrow R = TABLA(condiciones)$

Resulta una tabla que cumple las condiciones de TABLA.

Proyección  $\rightarrow R = TABLA[atributo1, atributo2]$

Resulta una tabla que contiene solo los atributos seleccionados de TABLA.

Combinación  $\rightarrow R = TABLA1[condiciones]TABLA2$

Resulta una combinación entre TABLA1 y TABLA2 cumpliendo las condiciones dadas, útil cuando se quiere obtener una tabla a base de hacer combinaciones de la misma pero modificada.

Secuencias de operaciones de algebra relacional ejemplo:

$A = TABLA1(condición)$

$B = A\{atributo1 \rightarrow atributoA\}$

$C = TABLA2 * B$

$R = C[atributo1, atributo2]$

## Tema 3.2 – Lenguajes: Álgebra relacional

**CREATE TABLE** <nombre\_tabla>

(<nombre\_columna> <tipo\_dato> [<restricciones\_col>] [<val\_por\_defecto>]

[, <nombre\_columna> <tipo\_dato> [<restricciones\_col>] [<val\_por\_defecto>]...]

[<restricciones\_tabla>]);  Ej: int, string ...

**INSERT INTO** <nombre\_tabla> [(<columnas>)]

( **VALUES** {<valor1> | **NULL**}, ..., {<valorN> | **NULL**} ) | <consulta> ;


**DELETE FROM** <tabla>

**WHERE** <condiciones>;

**UPDATE** <tabla>

**SET** <col> = {expresión/ **NULL**} [, <col> = {expresión/ **NULL**}...]


**WHERE** <condiciones> ;

 No repite atributo (Solo poner si se repiten las consultas)

**SELECT** [**DISTINCT** | **ALL**] <columnas\_que\_seleccionar> [, <funciones\_de\_agregación>]

**FROM** <tabla\_que\_consultar>

[ **WHERE** <condiciones> ]

**ORDER BY** <columna> [ **DESC** | **ASC** ], ... 

Si tienen el mismo primer atributo repetido ordena según el criterio seleccionado

**GROUP BY** <columnas\_según\_las\_que\_agrupar>

**HAVING** <condiciones\_para\_grupos>;

val\_por\_defecto: **DEFAULT** {<literal> | **NULL**}

restricciones\_col:

- **UNIQUE**: la columna no puede tener valores repetidos
- **PRIMARY KEY**: la columna es la clave primaria de la tabla
- **REFERENCES** <tabla> [<col>]: columna foreign key que referencia a la tabla indicada
- **CHECK** (<condiciones>): columna que debe cumplir las condiciones especificadas
- **NOT NULL**: la columna no puede tener valores nulos

restricciones\_tabla:

- **UNIQUE** (<cols>)
- **PRIMARY KEY** (<cols>)
- **FOREIGN KEY** (<cols>) **REFERENCES** <tabla> [<cols>]
- **CHECK** (<condiciones>): La table debe cumplir las condiciones especificadas, la condición puede referirse a una o más columnas de la tabla

Operadores:

- aritméticos: **\***, **+**, **-**, **/** → **SELECT(\*)** selecciona todo
- de comparación: **=**, **<**, **>**, **<=**, **>=**, **<>** → (diferente)
- lógicos: **NOT**, **AND**, **OR**
- otros:
  - <columna> **BETWEEN** <límite1> **AND** <límite2>
  - <columna> **IN** (<valor1>, <valor2> [...], <valorN>)]
  - <columna> **LIKE** <característica>
  - <columna> **IS** [NOT] **NULL**

funciones\_de\_agregación:

- **COUNT**(\*)
- **COUNT**(DISTINCT <columna>)
- **COUNT**(<columna>)
- **SUM**(expresión)
- **MIN**(expresión)
- **MAX**(expresión)
- **AVG**(expresión)

Sustitutos del **WHERE**

```
SELECT e.num_empl, p.num_proj, p.nom_proj
FROM empleats e, projectes p
WHERE e.num_proj = p.num_proj;
```

```
SELECT e.num_empl, p.num_proj, p.nom_proj
FROM empleats e INNER JOIN projectes p ON e.num_proj = p.num_proj;
```

```
SELECT e.num_empl, p.num_proj, p.nom_proj
FROM empleats e NATURAL INNER JOIN projectes p;
```

**UNION** → Tiene como a resultado la unión de dos selecciones diferentes y no repite resultados, no se necesita poner **DISTINCT**

Diferencia:

**WHERE ... NOT IN** (selección) → Da el valor del atributo si no está en la selección (subconsulta)

**WHERE NOT EXISTS** (selección) → Da el valor del atributo si no existe en la selección

## Tema 4 – Componentes lógicos de una base de datos

**Esquemas**: sirven para centralizar tareas administrativas como encender, apagar o otorgar privilegios de un conjunto de componentes lógicos a un usuario

```
CREATE SCHEMA [[nombre_catalogo]nombre_esquema] [AUTHORIZATION id_usuario]
[lista_elementos_esquema];
```

```
DROP SCHEMA nombre_esquema [RESTRICT | CASCADE]
```

- **RESTRICT**: borra un esquema solo si este está completamente vacío.
- **CASCADE**: borra un esquema, aunque contenga elementos.

**Aserciones:** restricciones de integridad que afectan a más de una tabla

`CREATE ASSERTION nombre CHECK (condición);`

**Vistas:** es una tabla virtual cuyo contenido está definido por una consulta

`CREATE VIEW nombre_vista AS SELECT (nombre_columna, ...)  
FROM nombre_tabla  
WHERE (condición)  
[WITH CHECK OPTION];` → No deja actualizar la vista

Actualizar vista → `UPDATE nombre_vista SET ... WHERE ...;  
SELECT * FROM nombre_vista;`

**Privilegios** → Conceden y quitan autorización de hacer según que cosas en una base de datos.

`GRANT privilegios ON objetos TO usuarios [WITH GRANT OPTION];` Da permiso para dar y quitar permisos que ese mismo usuario tiene

`REVOKE [GRANT OPTION FOR] privilegios ON objetos FROM usuarios {CASCADE | RESTRICT}` Quita el permiso de dar permiso

- **CASCADE:** Se revocan todos los privilegios concedidos por ese usuario
- **RESTRICT:** No se revoca ningún otro privilegio

Privilegios que se pueden dar sobre una vista (para conceder privilegios con SELECT) o una tabla: **SELECT, INSERT, UPDATE, DELETE.**

**ROLS** → Es una agrupación de privilegios definida para un grupo de usuarios específicos.

`CREATE ROLE nombre_role -- Crea ROLE  
GRANT privilegios ON objetos TO nombre_role -- Asigna privilegios del ROLE  
GRANT nombre_role TO usuario -- Dar privilegios a un usuario del ROLE`

## Tema 7 – Introducción al diseño de bases de datos relacionales

**Asociaciones:** es la representación de una relación entre dos o más objetos

**Asociaciones binarias:**



**Transformación de asociaciones binarias: caso uno a muchos (0 .. 1 a \*)** → Añadir la foreign key a la relación que corresponde a la clase del extremo “muchos” (\*).

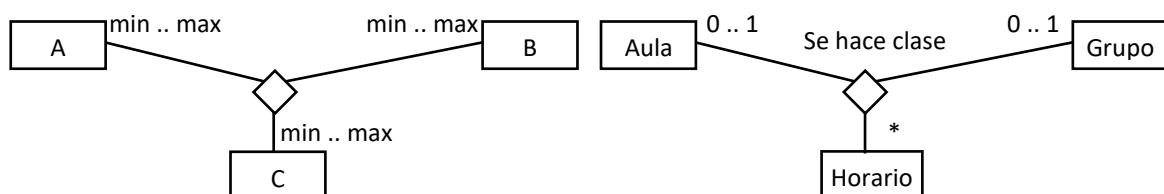
$A(\underline{\text{atributo}_a \text{ primary key}}, \dots)$   
 $B(\underline{\text{atributo}_b \text{ primary key}}, \dots, \text{atributo\_ref})$  mismo nombre  
 donde {atributo\_ref} referencia A

**Transformación de asociaciones binarias: caso uno a uno (0 .. 1 a 0 .. 1)** → Añadir a cualquiera de las dos relaciones una foreign key que referencie a la otra.

$A(\underline{\text{atributo}_a \text{ primary key}}, \dots)$   
 $B(\underline{\text{atributo}_b \text{ primary key}}, \dots, \text{atributo\_ref})$   
 $\text{NUEVA\_RELACION}(\underline{\text{atributo}_a \text{ primary key}}, \underline{\text{atributo}_b \text{ primary key}})$   
 donde {atributo\_a\_primary\_key} referencia A y {atributo\_b\_primary\_key} referencia B

**Transformación de asociaciones binarias: caso muchos a muchos (\* a \*)** → Se define una nueva relación donde se referencian las claves primarias de ambas relaciones

**Asociaciones ternarias:**



$A(\underline{\text{atributo}_a \text{ primary key}}, \dots)$   
 $B(\underline{\text{atributo}_b \text{ primary key}}, \dots)$

$C(\text{atributo}_c\_primary\_key, \dots)$

NUEVA\_RELACION(atributo<sub>a</sub>\_primary\_key, atributo<sub>b</sub>\_primary\_key, atributo<sub>c</sub>\_primary\_key)

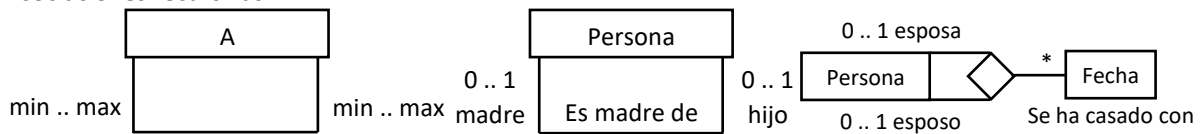
donde {atributo<sub>a</sub>\_primary\_key} referencia A,  
 {atributo<sub>b</sub>\_primary\_key} referencia B y  
 {atributo<sub>c</sub>\_primary\_key} referencia C

**Transformaciones de asociaciones ternarias: muchos-muchos-muchos** → La foreign key de la nueva relación está formada por las claves de las tres clases.

**Transformaciones de asociaciones ternarias: muchos-muchos-uno** → La foreign key de la nueva relación está formada por las claves de las dos clases de los “muchos” (\*) de la asociación.

**Transformaciones de asociaciones ternarias: muchos-uno-uno** → La foreign key de la nueva relación está formada por la clave de “muchos” y una de las otras dos claves (escoger la que se quiera).

**Asociaciones recursivas:**



**Transformaciones de asociaciones recursivas**

PERSONA(codigo\_persona, ..., codigo\_madre)  
 donde {codigo\_madre} referencia PERSONA

PERSONA(cod\_pers, ...)

FECHA(fecha)

CASAMIENTO(cod\_esposo, cod\_esposa, fecha)

o

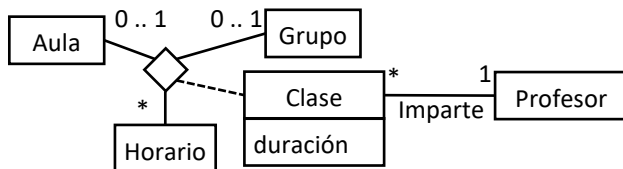
CASAMIENTO(cod\_esposo, cod\_esposa, fecha)

donde {cod\_esposo} referencia PERSONA,

donde {cod\_esposa} referencia PERSONA y

donde {fecha} referencia FECHA,

**Asociaciones de clases asociativas**



**Transformación de clases asociativas**

HORARIO(dia-sem, hora)

AULA(cod\_aula, ...)

GRUPO(gr, ...)

PROF(prof, ...)

CLASE(dia-sem, hora, cod\_aula, gr, duracion, prof)

o

CLASE(dia-sem, hora, cod\_aula, gr, duracion, prof)

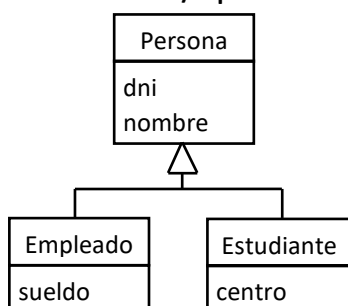
donde {dia-sem, hora} referencia HORARIO,

{cod\_aula} referencia AULA,

{gr} referencia GRUPO y

{prof} referencia PROFESOR

**Generalización/especificación**



**Transformación de la generalización/especialización**

PERSONA(dni, nombre)

EMPLEADO(dni, sueldo)

donde {dni} referencia PERSONA

ESTUDIANTE(dni centro)

donde {dni} referencia PERSONA

La **multiplicidad** (min .. max) de clase dentro de una asociación indica cuántas instancias de esa clase pueden asociarse con las otras clases de la asociación.

- 0 .. 1 → Sin instancia o sólo una (opcional)
- 1 → Una y siempre una instancia (obligatorio)
- \* → Cero o múltiples instancias (opcional)
- 1 .. \* → Múltiples instancias, pero al menos una (obligatorio)

## Tema 5 – Procedimientos y Disparadores

```
-- PROCEDIMIENTO QUE DEVUELVE UNA ÚNICA TUPLA
CREATE FUNCTION nombre_procedimiento(param_entrada tipus)
RETURNS tipo_retorno AS $$
DECLARE
  -- Variables que declarar, si no se inicializan por defecto son NULL
  nombre_variable [CONSTANT] tipo [NOT NULL] [{DEFAULT | :=} expression];
  -- Especifica el tipo de variable manualmente
  variable_retorno tipo_var;
  /* Especifica el tipo de variable idéntico al de un determinado atributo de una
  tabla */
  var tabla.atributo%TYPE;

BEGIN
  -- Sentencia de asignación
  nombre_variable := (SELECT atr FROM tabla WHERE condicion);

  -- Asignamos tuplas de un atributo a una variable
  SELECT atributo INTO variable_retorno
  FROM tabla
  WHERE condicion;

  RETURN variable_retorno;
END;
$$LANGUAGE plpgsql;
```

```
-- PROCEDIMIENTO QUE DEVUELVE UN CONJUNTO DE TUPLAS
CREATE TYPE conjunto AS (
  var1 tipo_var,
  var2 tipo_var,
  var3 tipo_var);

-- OPCION 1
-- El SETOF devuelve un conjunto de tuplas especificadas en el CREATE TYPE
CREATE FUNCTION nom_proc(param_entrada tipus) RETURNS SETOF conjunto AS $$
...
/* Va devolviendo a cada ejecución los valores de la variable.
El procedimiento acaba cuando se ejecuta un RETURN sin NEXT o llega al final */
RETURN NEXT variable;
...
END;
$$LANGUAGE plpgsql;

-- OPCION 2
CREATE FUNCTION nom_proc(param_entrada tipus) RETURNS conjunto AS $$
DECLARE
  conj conjunto;
  var tipo_var;

BEGIN
  ...
  conj.var1 := var;
  ...

  RETURN conj;
END;
$$LANGUAGE plpgsql;
```

```
-- SENTENCIAS CONDICIONALES
IF (condicion) THEN Bloque de sentencias;
ELSE IF (condicion) THEN Bloque de sentencias;
ELSE THEN Bloque de sentencias;
END IF;
```

```

/* - FOUND es un booleano con valor inicial False, pero cambia a true si
   en un SELECT ... INTO ... encuentra una tupla
   - UPDATE, INSERT o DELETE, FOUND cambia a true si almenos una fila se ve
   afectada por la sentencia
   - En una sentencia FOR, FOUND cambia a true al acabar el FOR se ha iterado
   alguna vez */
IF FOUND THEN Bloque de sentencias;

```

```

-- SENTENCIAS ITERATIVAS
-- Por cada statement ; al final
FOR var IN sentenciaSQL
LOOP statements END LOOP;

WHILE sentenciaSQL LOOP statements END LOOP;

```

```

-- GESTIÓN DE ERRORES
CREATE FUNCTION proc() RETURNS tipo_var AS $$
DECLARE
    missatge missatgesExcepcions.texte%TYPE;
BEGIN
    IF (condicion_error) THEN
        SELECT texte INTO missatge FROM missatgesExcepcions WHERE num = 1;
        RAISE EXCEPTION '%', missatge;
    END IF;

EXCEPTION
    WHEN raise_exception THEN RAISE EXCEPTION '%', SQLERRM;
    WHEN others THEN -- Gestiona otros errores
        SELECT texte INTO missatge FROM missatgesExcepcions WHERE num = 2;
        RAISE EXCEPTION '%', missatge;
END;
$$ LANGUAGE plpgsql;

```

```

-- DISPARADORES
CREATE FUNCTION nomFunc() RETURNS trigger AS $$
DECLARE
    var var_type;
BEGIN
    -- Ejemplo de cómo usar NEW (válido para OLD) para consultar el nuevo/viejo atributo
    SELECT * INTO var FROM table WHERE atr = NEW.atr1;
    ...
    RETURN {NEW | OLD | NULL};
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER nombre {BEFORE | AFTER} {INSERT | DELETE | UPDATE [OF columna, ...]}
ON tabla [FOR [EACH] {ROW | STATEMENT}]
EXECUTE PROCEDURE nomFunc;
/* FOR EACH ROW -> Se ejecuta para cada tupla de una tabla en una sentencia SQL
   FOR EACH STATEMENT -> Se ejecuta 1 vez por cada sentencia SQL */

```

	ROW	STATEMENT
BEFORE	RETURN NEW → INSERT, UPDATE → Se ejecuta la sentencia con el valor modificado	RETURN NULL → Ignoramos el valor devuelto
	RETURN OLD → DELETE, UPDATE (sin el cambio hecho en la sentencia)	
	RETURN NULL → No ejecuta la sentencia que activa el disparador	
AFTER	RETURN NULL → Ignoramos el valor devuelto	RETURN NULL → Ignoramos el valor devuelto



## Tema 8 – Transacciones y concurrencia

El objetivo de los SGBD es permitir un acceso simultáneo de múltiples usuarios a la misma BD preservando su integridad. Para ello se usan transacciones, estas son un conjunto de operaciones de lectura y/o actualización de la BD que acaban confirmando o cancelando los cambios hechos.

**Interferencias entre transacciones:** se producen cuando las transacciones no se aíslan adecuadamente entre sí.

<b>Actualización perdida:</b> se produce cuando intentamos escribir/actualizar mediante una operación, pero esta no se produce.	<b>Lectura no confirmada:</b> se produce cuando una transacción lee un dato que se ha modificado por otra transacción que después aborta.	<b>Lectura no repetible:</b> se produce si una transacción lee 2 veces el mismo dato y obtiene valores diferentes, debido a una modificación efectuada por otra transacción.	<b>Análisis inconsistente:</b> cuando a mitad de una ejecución de una transacción, otra transacción cambia el estado de una variable.	<b>Fantasma:</b> una transacción lee un conjunto de datos relacionado y existe otra transacción que añade nuevos datos que pertenecen a ese conjunto.
<p>T1: RU(A), W(A), COMMIT</p> <p>T2: RU(A), W(A), COMMIT</p>	<p>T1: RU(A), W(A), COMMIT</p> <p>T2: RU(A), W(A), ABORT</p>	<p>T1: R(A), R(A), COMMIT</p> <p>T2: RU(A), W(A), COMMIT</p>	<p>T1: R(A), R(B), COMMIT</p> <p>T2: RU(B), W(B), RU(A), W(A), COMMIT</p>	<p>T1: R(IC), R(C1), R(C2), R(C3)</p> <p>T2: RU(C3), W(C3), RU(IC), W(IC)</p>
<p>T1 → A → T2</p> <p>T2 → A → T1</p>	<p>T1 → A → T2</p> <p>T2 → A → T1</p>	<p>T1 → A → T2</p> <p>T2 → A → T1</p>	<p>T1 → A → T2</p> <p>T2 → B → T1</p>	<p>T1 → IC → T2</p> <p>T2 → IC, C3 → T1</p>
READ UNCOMMITTED READ COMMITTED REPEATABLE READ SERIALIZABLE	READ UNCOMMITTED READ COMMITTED REPEATABLE READ SERIALIZABLE	READ UNCOMMITTED READ COMMITTED REPEATABLE READ SERIALIZABLE	READ UNCOMMITTED READ COMMITTED REPEATABLE READ SERIALIZABLE	READ UNCOMMITTED READ COMMITTED REPEATABLE READ SERIALIZABLE

Las situaciones de conflicto aparecen cuando: 2 operaciones actúan sobre el mismo gránulo y **al menos** 1 de las operaciones es de **escritura** y la flecha va desde el primero que se ejecuta al segundo.

La **teoría de la serializabilidad** define las condiciones que se han de cumplir para que las transacciones estén aisladas entre sí correctamente.

- **Gránulos:** las transacciones están formadas por acciones (operaciones) sobre datos elementales llamados gránulos (página (bloque) del disco, un registro (una tupla), etc). Operaciones sobre gránulos:
  - R(G): Lectura del gránulo G. (Sentencia SQL: SELECT)
  - RU(G): Lectura con intención de modificación posterior del gránulo G. } Sentencias SQL: INSERT/UPDATE/DELETE
  - W(G): Escritura del gránulo G.
- **Horario o historia:** dónde se guarda el orden de las acciones dentro de cada transacción.
- **Horario serial:** cuando las transacciones se ejecutan completamente una detrás de otra sin solaparse.
- **Acciones conflictivas o no conmutables:** dos acciones serán conflictivas cuando 2 transacciones distintas operan sobre el mismo gránulo y al menos una de las acciones es de escritura.
- **Grafo de precedencias**
- **Horario serializable:** es un horario serializable aquel que al construir el grafo no tiene ciclos.

**Técnicas de control de concurrencia:** permite resolver las interferencias entre transacciones de dos maneras:

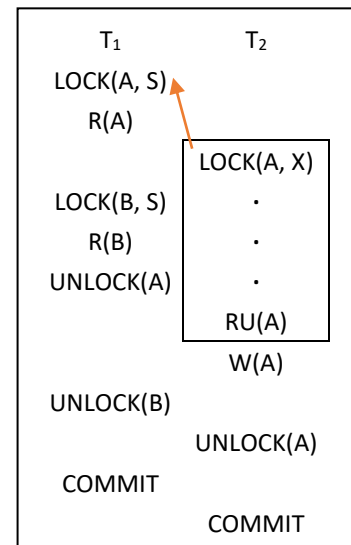
- Cancelando automáticamente las transacciones problemáticas y deshaciendo sus cambios.
- Suspendiendo la ejecución temporalmente y volviéndola a ejecutar cuando no haya peligro de interferencia.

**Control de concurrencia con reservas:** se reserva un gránulo con una cierta modalidad antes de efectuar una acción.

- LOCK(G, S): permite al gránulo G lecturas (modalidad compartida)
- LOCK(G, X): permite al gránulo G hacer lecturas y escrituras (modalidad exclusiva: ninguna otra transacción pueda hacer un LOCK del gránulo G hasta que este no se libere).
- UNLOCK(G): liberamos la reserva sobre el gránulo G.

En ciertos momentos es conveniente baja el **nivel de aislamiento** y dejar que se produzcan interferencias:

- READ UNCOMMITTED: evita que cualquier otra transacción se actualice hasta que acabe la transacción. Se hacen reservas de escritura (X) hasta el final de la transacción y no se hacen reservas de lectura (S).
- READ COMMITTED: impide que otra transacción lea datos que aún no se han confirmado. Se hacen reservas de escritura (X) hasta el final y se desbloquean las reservas de lectura (S) después de esta.
- REPEATABLE READ: hasta que no acabe la transacción, impide que otra transacción actualice un dato ya leído. Se hacen las reservas de escritura (X) y lectura (S) hasta el final.
- SERIALIZABLE: aislamiento total excepto fantasmas. Se hacen las reservas de escritura (X) y lectura (S) hasta el final.



**Recuperación:**

- Restauración: garantiza que la base de datos cumplan la atomicidad de las transacciones.
- Reconstrucción: recuperan el estado de la BD cuando se produce un pérdida total o parcial.
  - Reconstrucción hacia delante: implica deshacer los cambios de una transacción abortada.
  - Reconstrucción hacia atrás: implica rehacer los cambios de una transacción confirmada.

## Tema 9 – Almacenamiento y métodos de acceso

Hay 2 tipos de memoria:

- Memoria externa: es más lenta, pero más barata.
- Memoria interna: más rápida, pero más cara, es volátil y su capacidad es reducida.

Dado que no existe un estándar para los ficheros de las bases de datos, aprenderemos el patrón que siguen. La arquitectura de una BD tiene tres niveles, que sirven para comprender la relación entre los datos tal y como los ve el programador o el usuario final y tal y como están almacenadas.

- **Nivel físico:** los datos se almacenan en discos magnéticos controlados por el SO, que es el que realmente efectúa las lecturas y escrituras, ahora bien, es la SGBD la que decide cuando hacer estas operaciones y el que conoce como están físicamente estructurados los datos y como interpretarlos.

**Ficheros:** es la unidad global a partir de la cual el SO gestiona los datos en los discos magnéticos. Es un conjunto de extensiones.

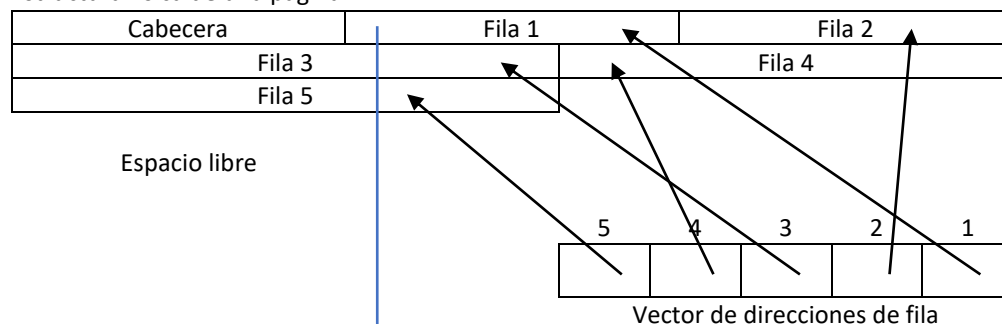
**Extensión:** es la unidad de adquisición de espacio por cada fichero. (es un múltiplo entero de la página). Es el nombre de páginas consecutivas que el SO adquiere a petición de la SGBD cuando esta detecta que necesita más espacio para un fichero determinado

**Página:** es el componente físico más pequeño, contiene y almacena los datos a nivel lógico. Hay 2 formas de ver el concepto de página:

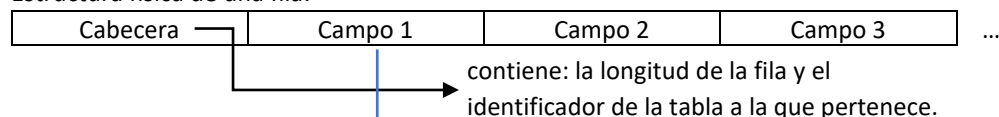
- **Unidad discreta de transporte de datos (E/S)** entre la memoria externa (disco) y la memoria interna. En SO normalmente bloque.
- **Unidad de organización de datos almacenados.** El espacio de disco se estructura en un nombre múltiplo de páginas, y cada página se puede direccionar individualmente.

La estructura física sigue una longitud fija (2K, 4K, 8K)

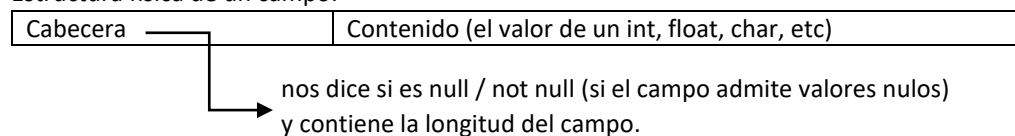
Estructura física de una página:



Estructura física de una fila:



Estructura física de un campo:



- **Nivel virtual:** hace de intermediario entre el nivel físico y el lógico, ya que:
  - Si las tablas son muy grandes → Hay fragmentos en distintos dispositivos → Se asocia un fragmento a un fichero diferente.
  - Si las tablas son muy pequeñas → Hay que agruparlas en un fichero.

- Por los objetos grandes, que se almacenan separadamente.
- Por los índices, disparadores, restricciones...

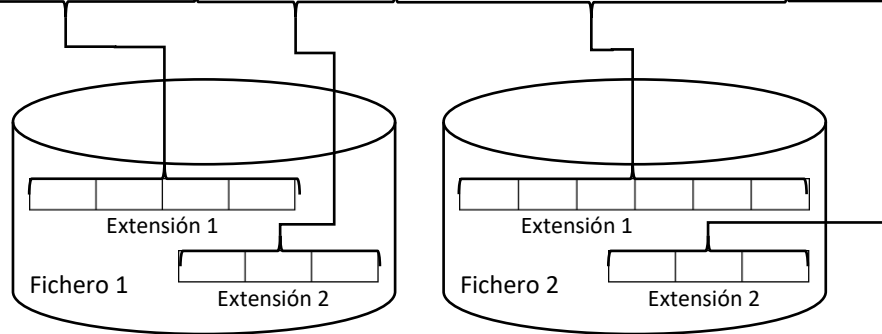
A nivel virtual también existe un sistema de paginación que se hace de manera paralela a la física.

Correspondencia entre páginas reales y páginas virtuales:

a. Nivel virtual

Núm. página	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
-------------	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

b. Nivel físico



- Nivel lógico

Los **métodos de acceso** nos permiten la lectura o actualización de una tabla, dado que cuando leemos o actualizamos un dato accedemos a la paginación virtual y de esta a la física. Los siguientes métodos de acceso:

- **Acceso por posición:**

- Directo

INSERT INTO x VALUES (a, b, c, d);

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

↑

- Secuencial

SELECT \* FROM x

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑

- **Acceso por valor:**

- Directo
  - `SELECT * FROM tabla WHERE atr = x;`
  - `DELETE FROM tabla WHERE atr = x;`
  - `UPDATE tabla SET atr2 = y WHERE atr1 = x;`
- Secuencial
  - `SELECT * FROM tabla ORDER BY atr;`
  - `SELECT * FROM tabla WHERE atr > x AND atr < y;`
  - `UPDATE tabla SET atr2 = y WHERE atr1 = x;`
  - `DELETE FROM table WHERE atr > x AND atr < y;`

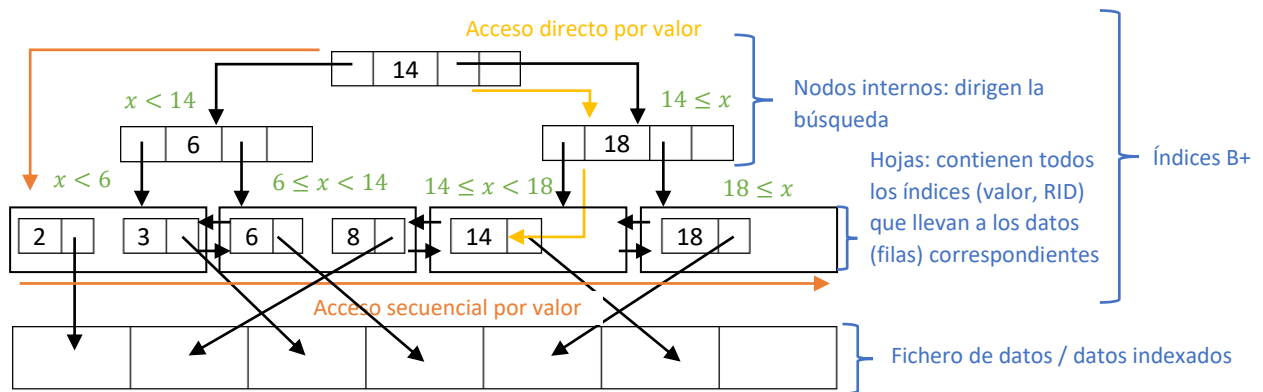
- **Acceso por diversos valores:** cuando se acceden a las filas por varios atributos

- `SELECT * FROM tabla ORDER BY atr1, atr2`
- `SELECT * FROM tabla WHERE atr1 = x AND atr2 < y;`

Se usan **índices** para una implementación más eficiente, porque ocupan menos espacio que los datos. Funcionan de manera similar a los índices de los libros. Y hacen el acceso por valor vía acceso por posición.

Índice			(valor, RID)			
Datos					fila	

**Árboles B<sup>+</sup>:** es una de las maneras de estructurar los índices. Un árbol de orden  $d$  ( $d$ -ario) tiene como mucho  $2d$  valores y  $2d + 1$  apuntadores. Ejemplo:



**mejor rendimiento:**

- Todos los nodos han de estar llenos como mínimo al 50%, menos la raíz.
- Todas las hojas han de estar al mismo nivel.

Cada nodo se almacena en una página → Para acceder a una hoja harán falta  $n$  niveles.

Ejercicio: quantas filas/registros se podrían indexar en un árbol de orden  $d$  y lleno al  $x$  %:

- Nivel 1: 1 nodo con  $x$  valores y  $(2d + 1) * x$  apuntadores.
  - Nivel 2:  $(2d + 1) * x$  nodos con  $2d * x$  valores y  $(2d + 1) * x$  apuntadores cada uno.
  - Nivel 3:  $((2d + 1) * x)^2$  nodos con  $2d * x$  valores y  $(2d + 1) * x$  apuntadores cada uno.
  - Nivel 4:  $((2d + 1) * x)^3$  nodos con  $2d * x$  entradas →  $((2d + 1) * x)^3 * 2d * x$  filas/entradas/registros.
- Para saber la altura de un árbol:  $\log_{(2d)*x+1} n^o \text{ filas/entradas/registros}$

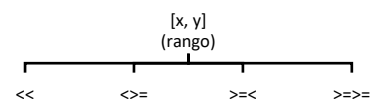
**Índices agrupados** (clústers): es aquel en que los datos que indexa están ordenados físicamente a partir del acceso secuencial por valor que proporciona. Esto supone que hay que mantener un orden físico, para ello se deja un % de espacio libre en las páginas, si se llenan se llenan páginas excedentes encadenadas y en el caso de haber un % elevado de éstas, se regroupan.

**Accesos por diversos valores: accesos directos**

- **Estrategia de intersección de RIDs:** uso de estructuras de árboles B+, para crear conjuntos con las condiciones y devuelve la intersección de los conjuntos.
- **Estrategia de índice multi-atributo:** uso de estructura de árboles B+, el resultado lo llamamos lista y establecemos un orden lineal entre las listas, de esta manera se ordenan según la primera condición, de las que cumplen la primera condición se ordenan según la segunda... Coste =  $h + F + R(\text{cond.})/f$

**Accesos por diversos valores: accesos secuenciales y mixtos:**

- Estrategia de índice multi-atributo
- **Estrategia de índice multi-atributo multi-dimensional:** no sigue un orden lineal



Creación de un índice	<code>CREATE INDEX nom_indice ON tabla(atr)</code>
Para un índice descendiente	<code>CREATE INDEX nom_indice ON tabla(atr DESC)</code>
Índice agrupado	<code>CREATE INDEX CLUSTER nom_indice ON tabla(atr)</code>
Índice sin valores repetidos	<code>CREATE UNIQUE INDEX nom_indice ON tabla(atr)</code>
Índice multi-atributo	<code>CREATE INDEX nom_indice ON tabla(atr1, atr2)</code>

**Coste acceso secuencial por valor:**

- **Índice árbol B+, no agrupado:**  
Coste = coste acceso índice + coste acceso fichero de datos =  $(h + F) + |R(a \geq X)|$
- **Índice árbol B+, agrupado:**  
Coste = coste acceso índice + coste acceso fichero de datos =  $h + D = h + [R(a \geq X)/f]$

**Coste acceso directo por valor:**

- **Índice árbol B+, agrupado/no agrupado, sin valores repetidos:**

Coste = coste acceso índice + coste acceso fichero de datos =  $h + 1$

- **Índice árbol B+ no agrupado, con posibilidad valores repetidos:**  
Coste = coste acceso índice + Coste acceso fichero de datos =  $(h + F) + |R(b = Y)|$
- **Índice árbol B+ agrupado, con posibilidad valores repetidos:**  
Coste = coste acceso índice + coste acceso fichero de datos =  $h + D = h + [R(b = Y)/f]$
- **Acceso por diversos valores: estrategia de intersección de RIDs:**  
Coste = coste acceso índice<sub>1</sub> + ... + coste acceso índice<sub>n</sub> coste acceso fichero de datos =  
=  $(h_a + F_a) + (h_b + F_b) + |R(a \geq X \wedge b = Y)|$

**Nota:** el coste de acceso a índice puede disminuir en 1 si la raíz está almacenada en memoria.

$h$  = niveles árbol

$F$  = número de hojas recorridas

$R(\text{condición})$  = número de tuplas que cumplen la condición en el fichero de datos

$D$  = nombre de páginas del fichero de datos que hay que recorrer

$f$  = número de registros por página / factor de bloqueo

## Tema 6 – Programación usando SQL

**SQL estático:** las sentencias SQL son fijas y las mismas en cualquier ejecución del programa.

- **SQL:** se trata de incrustar sentencias SQL estáticas dentro del programa, por ello hace falta una pre-compilación, traducir las sentencias SQL en sentencias del lenguaje de programación utilizado, por ello es poco portable, más difícil de programar pero más eficiente y ocupa menos líneas de código.

**SQL dinámico:** las sentencias SQL se incrustan durante la ejecución del programa.

- **JDBC:** es una API que define unos métodos de conexión para la conexión y acceso a datos remotos desde un programa escrito en Java. Las funciones SQL se compilan durante la ejecución, por lo que lo hace más portable y más fácil de programar pero menos eficiente y ocupa más líneas de código. Tipos de drivers:

```
import java.sql.*; // importamos las librerías sql

// Indicamos el driver JDBC que se usará para acceder a la base de datos
Class.forName("nombreDriver");

// Propiedades como el username y password al que queremos conectarnos
Properties props = new Properties();
props.setProperty("user", "miUsuario");
props.setProperty("password", "miContraseña");
// Conexión con la base de datos
Connection c = DriverManager.getConnection("url", props);
c.setAutoCommit(false);

// Selección del schema
set schema nombreSchema

/* Comunicación con la base de datos */
//-----
// Comunicación usando Statement (se usa cuando ejecutamos el código 1 sola vez)
Statement s = c.createStatement();
// Consultas
nt consulta = s.executeQuery("Sentencia de consulta SQL");
// Modificaciones
int modificaciones = s.executeUpdate("Sentencia insert/delete/update SQL");
if (modificaciones == 0) System.out.println("Se ha modificado con éxito");
s.close(); // Cerramos la conexión con la base de datos
//-----
// Comunicación usando PreparedStatement
// (se usa cuando ejecutamos el código más de 1 vez, como por ejemplo en bucles)
// Consultas
PreparedStatement ps = c.prepareStatement("Sentencia consulta SQL");
ps.executeQuery();
// Modificaciones
PreparedStatement ps = c.prepareStatement("Sentencia insert/delete/update SQL");
```

```

int modificaciones = ps.executeUpdate();
if (modificaciones == 0) System.out.println("Se ha modificado con éxito");
ps.close(); // Cerramos la conexión con la base de datos

// ResultSet



| telefono |
|----------|
|          |
|          |
|          |


// siguiente = r.next(); // siguiente = true
// siguiente = r.next(); // siguiente = true
// siguiente = r.next(); // siguiente = false

ResultSet r = s.executeQuery("Consulta SQL");
r.next(); // indica si existe otra tupla a la que acceder
// Ejemplo
String telf = r.getString("telefono");
if (telf == null) ...; // Para saber si el valor es nulo
if (telf.isNull()) ...; // Otra manera para saber si el valor es nulo

// Transacciones
c.commit(); // Guarda permanentemente los cambios hechos
c.rollback(); // deshace los cambios que no se hayan ya guardado en la BD

/* Excepciones (así es como se crea una clase para programar con SQL) */
public class progrEx1 {
    public static void main (String[] args) {
        try {
            // Enregistrar el driver
            // Conexión a la base de datos
            // Consultes-modificaciones
            // Desconexión
        }
        catch (ClassNotFoundException ce) {
            System.out.println ("Error al cargar el driver d'informix.");
        }
        catch (SQLException se) {
            System.out.println ("Error conexión o acceso a la base de
datos");}
    }
}

// Gestión de errores
try {
    // Ejecutar el Statement o PreparedStatement
} catch (SQLException se) {
    if (se.getSQLState().equals("Código")) System.out.println("Error que sea");
    else System.out.println(se.getSQLState() + " " + se.getMessage())

```