

Tema 7: Definición de Datos

Un lenguaje de BD completo se compone de comandos organizados en

- **LDD:** Lenguaje de definición de datos: ordenar para crear, modificar o eliminar tablas, visores...

CREATE , ALTER , DROP , ..

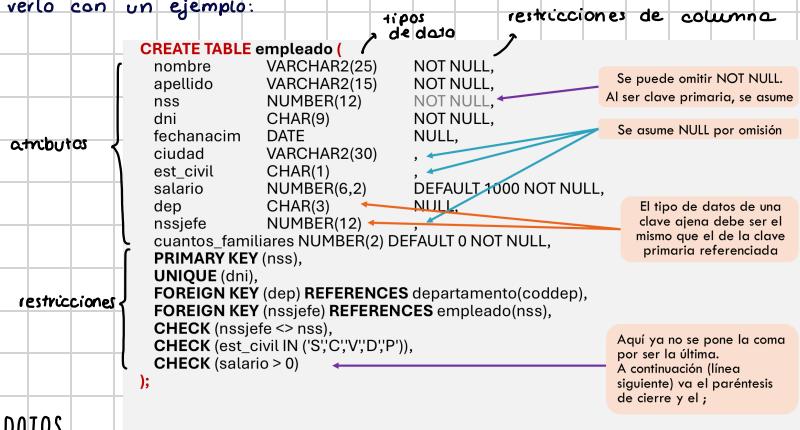
- LMD: Lenguaje de manipulación de datos: introducir datos, actualizarlos, eliminarlos...
INSERT, UPDATE, DELETE y SELECT

Sentencia CREATE TABLE

Sirve para crear tablas (relación en el Modelo-Relacional). El nombre de la tabla será único y dentro habrá que indicar:

- Columnas (atributos): con su nombre, tipo de datos y restricciones
 - Restricciones de tabla: clave primaria, ajena, restricciones...

Vamos a verlo con un ejemplo:



TIPOS DE DATOS

- ## • Cadenas de caracteres:

`CHAR[size]` -- por defecto `size = 1`

VARCHAR2(size)

- ## • Números:

NUMBER[(p[s])] -- por defecto p = 38 y s = C

- Fecha :

DATE

RESTRICCIONES DE COLUMNAS

- Cláusula **MNULL o NOT NULL**: por omisión se asume **NULL**, excepto para las columnas componentes de una clave primaria, que nunca podrán aceptar **NULL**.
- Cláusula **DEFAULT valor**: debe ir dentro del tipo de datos y delante de cualquier otra restricción.

RESTRICCIONES DE TABLA

- Cláusula **PRIMARY KEY (lista_columnas)**: indica que columnas forman parte de la clave primaria, asegura que no puedan tener el valor **NULL**.
- Cláusula **UNIQUE KEY (Lista_columnas)**: indica que columnas forman parte de la clave alternativa. Puede haber muchas y puede ser compuesta.
- Cláusula **CHECK (expresión)**: para especificar restricciones adicionales, que deben cumplir las columnas indicadas. Debe ser una expresión booleana. sólo se pueden comparar columnas de la tabla o columnas y constantes.
- Cláusula **FOREIGN KEY (Lista_columnas)**: define que columnas forman una clave ajena. Se indica la tabla y la clave a la que referencia. Debe coincidir el tipo de datos y ser compuesta en caso de que la clave a la que refiera sea. !!

Ejemplo de clave ajena compuesta:

```
CREATE TABLE hotel(  
    codigo CHAR(4)      NOT NULL,  
    nombre VARCHAR2(30) NOT NULL,  
    ... -- otras columnas  
    CONSTRAINT hotel_pk PRIMARY KEY(codigo)  
);  
  
CREATE TABLE salon_hotel(  
    id_salon CHAR(2)    NOT NULL,  
    hotel     CHAR(4)    NOT NULL,  
    capacidad NUMBER(3) NOT NULL,  
    ... -- otras columnas  
    CONSTRAINT salon_pk PRIMARY KEY (hotel, id_salon),  
    CONSTRAINT salon_fk_hotel FOREIGN KEY(hotel) REFERENCES hotel(codigo)  
    ...  
);  
Cada salón referencia al hotel en el  
cual está ubicado (clave ajena)
```

```
CREATE TABLE reserva_salon(  
    numero NUMBER(3) NOT NULL,  
    hotel   CHAR(4)  NOT NULL,  
    salon   CHAR(2)  NOT NULL,  
    fecha   DATE     NOT NULL,  
    ... --otras columnas  
    CONSTRAINT reserva_pk PRIMARY KEY(numero),  
    CONSTRAINT res_fk_salon FOREIGN KEY(hotel,salon)  
        REFERENCES salon_hotel(hotel,id_salon)
```

La clave primaria está
compuesta por dos columnas
);

Una clave ajena
debe tener tantas
columnas como
tiene la clave
primaria a la que
referencia

Hay operaciones que pueden romper la integridad Referencial, por ejemplo, cambiar una clave primaria que está siendo referenciada en otra tabla, o borrar una fila que está siendo referenciada.

CLIENTE			
codigo	nombre	direccion	ciudad
1210	Garcia, A.	Gran Via, 6	Murcia
0300	Zapata, D.	Ronda Norte, 3	Murcia
1003	Argon, C.	Paseo Rosales, 9	Molina
2689	Sancho, B.	Plaza Mayor, 2	Patijo
3679	Burgos, C.	Camino Viejo, 20	Yedla

CUENTA		
número	saldo	titular
200	85.005	2689
505	40.000	1003
821	50.000	1210
426	35.620	1003
005	29.872	2689
315	3.500	0300

* Si se borra la fila que contiene la clave primaria a la que referencia esa clave ajena.

¿Cómo evitamos que rompa la integridad referencial?

Debemos definir las acciones de mantenimiento, es decir, como actuara si se borra la FOREIGN KEY (ON DELETE) o si se actualiza (ON UPDATE).

Acciones que se pueden realizar:

ON DELETE

- NO ACTION: rechaza la acción
- CASCADE: propaga la eliminación
- SET NULL: establecer a nulo (si es posible)
- DEFAULT: valor por defecto (si existe)

ON UPDATE

- NO ACTION: rechaza la acción
- CASCADE: propaga la modificación
- SET NULL: establecer a nulo (si es posible)
- DEFAULT: valor por defecto (si existe)

Según el tipo de situación y usando el sentido común, elegimos la acción que se ajuste mejor (y sea posible).

CREATE TABLE empleado (

...,
 FOREIGN KEY (dep) REFERENCES departamento(coddep)
 ON DELETE NO ACTION —> no se permite borrar un departamento
 ON UPDATE CASCADE, —> se actualiza en cascada
 FOREIGN KEY (nssjefe) REFERENCES empleado(nss)
 ON DELETE SET NULL —> si se borra un empleado jefe a NULL
 ON UPDATE CASCADE, —> si se cambia se actualiza
 ...);

ANSI SQL

!!

En ORACLE solo se permiten 3 acciones de ON DELETE (NO ACTION, CASCADE y SET NULL) y NO EXISTE la cláusula ON UPDATE (se asume NO ACTION).

En las prácticas las escribiremos en forma de comentarios.

NOMBRES DE RESTRICCIÓN

Conviene dar nombre a las restricciones como las PRIMARY KEY, UNIQUE, CHECK...

Para ello se antepone a dichas cláusulas:

CONSTRAINT nombre_R1

El nombre debe ser único y podemos seguir un patrón:

abreviaturaTabla_TipoRestriccion

```

CREATE TABLE empleado (
    nombre          VARCHAR2(25)  NOT NULL,
    apellido        VARCHAR2(15)  NOT NULL,
    nss            NUMBER(12)   NOT NULL,
    dni             CHAR(9)      NOT NULL,
    fechanacim     DATE         NULL,
    ciudad          VARCHAR2(30)  ,
    est_civil       CHAR(1)      ,
    salario         NUMBER(6,2)   DEFAULT 1000 NOT NULL,
    dep             CHAR(3)      NULL,
    nssjefe         NUMBER(12)   ,
    cuantos_familiares NUMBER(2)  DEFAULT 0 NOT NULL,
    CONSTRAINT emp_pk PRIMARY KEY (nss),
    CONSTRAINT emp_ak UNIQUE (dni),
    CONSTRAINT emp_fk_dep FOREIGN KEY (dep)
        REFERENCES departamento(coddep),
        -- ON DELETE NO ACTION ON UPDATE CASCADE
    CONSTRAINT emp_fk_emp FOREIGN KEY (nssjefe) REFERENCES empleado(nss),
        -- ON DELETE NO ACTION ON UPDATE CASCADE
    CONSTRAINT emp_jefe_ok CHECK (nssjefe <> nss),
    CONSTRAINT emp_ec_ok CHECK (est_civil IN ('S','C','V','D','P')),
    CONSTRAINT emp_sal_ok CHECK (salario > 0)
);

```



Sentencia ALTER TABLE

Permite modificar la estructura de una tabla: añadir columnas, eliminarlas, modificarlas, añadir restricciones, eliminarlas, modificarlas.

- Añadir columna:

ALTER TABLE empleado se puede añadir la cláusula NULL (o NOT) y
ADD (fecha_contrato DATE); DEFAULT

- Renombrar columnas.

ALTER TABLE empleado
RENAME COLUMN fecha_contrato **TO** inicio_contrato;

- Renombrar tabla:

ALTER TABLE empleado **RENAME TO** emp; para renombrar un elemento:
RENAME _ TO _

- Modificar columna:

ALTER TABLE empleado se puede modificar el tipo,
MODIFY apellido **VARCHAR2(30);** el DEFAULT, el NULL, el UNIQUE...
--antes tamaño 15

- Eliminar columna / columnas:

ALTER TABLE empleado
DROP COLUMN fechanacim; / **ALTER TABLE** empleado
DROP (fechanacim, est_civil);

Si una columna está referenciada no se permitirá su eliminación. Se podrá forzar su eliminación con CASCADE CONSTRAINTS.

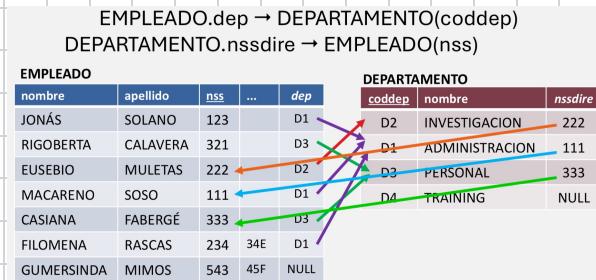
**ALTER TABLE departamento
DROP COLUMN coddep CASCADE CONSTRAINTS;**

La columna "dep" que hace referencia a "coddep", ya no sería clave ajena, si no un atributo normal.

- Añadir una restricción:

**ALTER TABLE tabla ADD
CONSTRAINT nombre_Rl definición_Rl;** ya sea CHECK, clave primaria, ajena...

CICLO REFERENCIAL



Si dos tablas se relacionan entre sí y queremos crearlas, primero creamos una sin definir la restricción clave ajena, luego la otra, y luego añadimos la restricción con ALTER TABLE

**ALTER TABLE empleado ADD CONSTRAINT emp_fk_dep
FOREIGN KEY dep REFERENCES departamento(coddep);
-- ON DELETE NO ACTION ON UPDATE CASCADE**

- Desactivar una restricción:

**ALTER TABLE tabla
DISABLE CONSTRAINT nombre_Rl;**

- Activar una restricción

**ALTER TABLE tabla
ENABLE CONSTRAINT nombre_Rl;**

ELIMINAR UNA RESTRICCIÓN

Para eliminar cualquier restricción usamos:

ALTER TABLE *tabla*
DROP CONSTRAINT *nombre_RI* [**CASCADE**];

Si la restricción es clave primaria podemos usar **DROP PRIMARY KEY [CASCADE]** en vez de la segunda línea. (o **DROP UNIQUE()** para claves alternativas)

Eliminar una clave falla si existe alguna clave ajena que la refiere, ahí entra en juego la opción **CASCADE**.

IMPORTANTE!! Eliminar la restricción clave primaria no es eliminar su columna, si no que ahora permite valores repetidos.

Sentencia **DROP TABLE**

Destrucción de una tabla (estructura y contenido). Si se omite **CASCADE CONSTRAINTS** solo destruye la tabla si no se le hace referencia desde otra.

DROP TABLE *table*
[**CASCADE CONSTRAINTS**] [**PURGE**];

Con **PURGE**, se elimina la tabla y se libera su espacio en un solo paso.

En el powerpoint del tema hay ejemplos de esquema lógico estándar para practicar, además de su implementación en SQL

Tema 8: manipulación de datos

Consulta

Permite extraer información sobre las tablas almacenadas.

Esto lo hacemos en Oracle mediante la sentencia `SELECT`:

```
SELECT lista_columnas  
FROM lista_tablas  
WHERE condición ; ← siempre termina con un punto y coma
```

Las condiciones pueden ser varias, unidas con `AND`, `OR` o incluso pueden ser negativas con `NOT`.

Para eliminar filas duplicadas se usa la cláusula `DISTINCT`, que va justo después de `SELECT`.

Además, para seleccionar todas las columnas de la tabla o tablas del `FROM`, podemos, o bien escribir el nombre de todas ellas, o usar `SELECT *`.

CADERAS DE CARACTERES

- Operador `LIKE`: sirve para comparar cadenas. Sigue usar comodines:

- % cantidad cualquiera de caracteres
- _ un solo carácter

* Nombres y apellidos de empleados de Las Torres de Cotillas o Cabezo de Torres
SELECT nombre, apellido
FROM empleado
WHERE ciudad LIKE '%TORRES%';

- Operador `||`: concatenación de cadenas

* Nombres completos, en una sola columna, de los empleados de las ciudades de Aledo, Alguazas, Alhama y Alcantarilla

```
SELECT nombre || ' ' || apellido  
FROM empleado  
WHERE ciudad LIKE 'AL%';
```

ARITMÉTICA

Se pueden aplicar operaciones (+, *, -, /) sobre las columnas para mostrarlo en el resultado del `SELECT`.

* ¿Cómo quedarían los salarios de los empleados del departamento D3 tras un aumento del 10%?

```
SELECT apellido, nombre, 1.1*salario  
FROM empleado  
WHERE dep = 'D3';
```

apellido	nombre	1.1*salario
CALAVERA	RIGOBERTA	990
FABERGÉ	CASIANA	1012

El valor de "salario" en empleado no cambiará. !!

FECHAS

- **TO_CHAR(fecha, formato)**: convierte una fecha en una cadena.
- **TO_DATE(cadena, formato)**: convierte una cadena en una fecha.

Formatos (ejemplo):

'dd/mm/yyyy'
'dd-Mon-yyyy'
'yyyy'
'dd/month hh24:mi:ss'

* Año de nacimiento y ciudad del empleado con apellido 'RASCAS'

SELECT TO_CHAR(fechanacim, 'yyyy'), ciudad
FROM empleado
WHERE apellido = 'RASCAS';

TO_CHAR(fechanacim, 'yyyy')	ciudad
1970	MURCIA

* Nombres y apellidos de los empleados nacidos antes de 1972

SELECT nombre, apellido
FROM empleado
WHERE fechanacim < TO_DATE('01/01/1972', 'dd/mm/yyyy');

- **EXTRACT(campo FROM fecha)** : campo puede ser YEAR, MONTH, DAY, HOUR, MINUTE, SECOND

CONJUNTOS DE VALORES

Se puede usar IN o

Nss de lxs empleadxs que tienen padres/madres o abuelxs a su cargo

SELECT nssemp

NOT IN.

FROM familiar

WHERE parentesco IN ('MADRE', 'PADRE', 'ABUELO', 'ABUELA');

NULL

Es una marca que indica desconocimiento y ausencia de información. No se puede comparar, null es distinto de cualquier otra cosa, incluso de otro null.

Para comprobar si una columna tiene el valor null, se necesita un **comparador específico**:

• IS NULL, IS NOT NULL

SELECT nombre, apellido
FROM empleado
WHERE nssjefe IS NULL;

ORDEN DE PRESENTACIÓN

Permite ordenar las filas con la cláusula ORDER BY. de manera ascendente ASC (por defecto) o descendente DESC.

SELECT apellido, nombre, fechanacim
FROM empleado
WHERE ciudad = 'MURCIA'
ORDER BY apellido ASC, nombre DESC;

REUNIÓN

¿Qué ocurre si queremos mostrar datos almacenados en varias tablas?

SELECT *
FROM empleado JOIN departamento
ON dep = coddep;

Unimos con JOIN y escribimos el vínculo entre las 2 tablas con ON.
↓
clave ajena y clave primaria

EMPLEADO				
nombre	apellido	nss	...	dep
JONAS	SOLANO	123	...	D1
RIGOBERTA	CALAVERA	321	...	D3
EUSEBIO	MULETAS	222	...	D2
MACARENO	SOSO	111	...	D1
CASIANA	FABERGÉ	333	...	D3
FILOMENA	RASCAS	234	...	D1
GUMERSINDA	IMIMOS	543	...	NULL

Resultado del JOIN: una tabla con una fila por cada una de las filas vinculadas de las tablas origen

nombre	apellido	nss	...	dep	nombre	coddep	nssdire
JONAS	SOLANO	123	...	D1	ADMINISTRACION	D1	111
RIGOBERTA	CALAVERA	321	...	D3	PERSONAL	D3	333
EUSEBIO	MULETAS	222	...	D2	INVESTIGACION	D2	222
MACARENO	SOSO	111	...	D1	ADMINISTRACION	D1	111
CASIANA	FABERGÉ	333	...	D3	PERSONAL	D3	333
FILOMENA	RASCAS	234	...	D1	ADMINISTRACION	D1	111

DEPARTAMENTO		
nombre	coddep	nssdire
INVESTIGACION	D2	222
ADMINISTRACION	D1	111
PERSONAL	D3	333
TRAINING	D4	NULL

SELECT *
FROM empleado JOIN departamento
ON dep = coddep;

ni el departamento D4 ni Gumersinda aparecen, porque Gumersinda tiene dep a NULL, y el D4 no está referenciado.

La condición de reunión puede incluir más de 1 comparación por igualdad, ligadas con un AND. Esto ocurre cuando la clave ajena y la clave primaria son compuestas.

SELECT ISBN, numero, estante, fecha
FROM prestamo JOIN ejemplar → reunión de dos tablas
ON libro = ISBN → condición de reunión
AND ejemplar = numero con 2 comparaciones
WHERE socio = 'S03'; → condición de selección de filas

Clave primaria EJEMPLAR: (ISBN, numero)

↓ ↓
Clave ajena PRÉSTAMO: (libro, ejemplar)

CALIFICACIÓN

Distintas tablas pueden tener columnas que se llamen igual. Esto puede causar ambigüedad en un SELECT si las seleccionamos.

La mejor opción es usar pseudónimos para diferenciar las tablas:

SELECT e.nombre, apellido, d.nombre
FROM empleado e JOIN departamento d
ON dep = coddep;

Solo hace falta calificar las columnas ambiguas, pero se puede hacer con todas.

RENOMBRAR COLUMNAS

SELECT e.nombre empleado, j.nombre jefe
FROM empleado e JOIN empleado j
ON e.nssjefe = j.nss;

Resultado SIN los nuevos nombres

e.nombre	j.nombre
JONAS	MACARENO
RIGOBERTA	CASIANA
EUSEBIO	JONAS
CASIANA	JONAS
FILOMENA	MACARENO

Resultado CON los nuevos nombres

empleado	jefe
JONAS	MACARENO
RIGOBERTA	CASIANA
EUSEBIO	JONAS
CASIANA	JONAS
FILOMENA	MACARENO

Simplemente añadimos el nuevo nombre separado por un espacio de la columna.

REUNIÓN PARTE II:

Se pueden usar varios JOIN OM en una SELECT para extraer información de varias tablas

* Para cada familiar 'ABUELA' o 'MADRE', mostrar tal parentesco, el apellido del empleado del que depende, y el nss del director del departamento al que pertenece el/la empleadox.

SELECT parentesco, apellido, nssdire
FROM familiar JOIN empleado
ON nssemp = nss
JOIN departamento
ON dep = coddep
WHERE parentesco IN ('ABUELA', 'MADRE');

El 1º JOIN obtiene una fila con los datos de cada familiar y los de su empleado

El 2º JOIN obtiene una fila para cada familiar con todos sus datos, los de su empleado y los del departamento al que pertenece el empleado

Antes, las tablas del JOIN se escribían en el FROM separadas por comas, y la condición de reunión iba en el WHERE.

notación clásica]

```
SELECT e.nombre, apellido, ciudad  
FROM empleado e, departamento d  
WHERE dep = coddep  
AND d.nombre = 'INVESTIGACION';  
← reunión de tablas  
← condición de reunión  
← condición de selección
```

Si se hace un JOIN de varias tablas y se omite el WHERE, se combina cada fila de una tabla con las filas de la otra: producto cartesiano

REUNIÓN NATURAL

El SGBD asume que la condición de reunión es las columnas que se llamen igual.

```
SELECT ...
```

```
FROM R1 NATURAL JOIN R2
```

```
WHERE ...
```

REUNIÓN EXTERNA

Necesaria si se necesita obtener filas con NULL en la condición de reunión o no referenciales.

LEFT [OUTER] JOIN

Reunión externa izquierda

RIGHT [OUTER] JOIN

Reunión externa derecha

FULL [OUTER] JOIN

Reunión externa total o completa

Ejemplo de reunión externa con la tabla departamento y empleado.

```
SELECT *  
FROM (empleado e FULL JOIN departamento d  
ON dep = coddep);
```

e.nombre	apellido	nss	...	dep	d.nombre	coddep	nssdire
JONÁS	SOLANO	123	...	D1	ADMINISTRACION	D1	111
RIGOBERTA	CALAVERA	321	...	D3	PERSONAL	D3	333
EUSEBIO	MULETAS	222	...	D2	INVESTIGACION	D2	222
MACARENO	SOSO	111	...	D1	ADMINISTRACION	D1	111
CASIANA	FABERGÉ	333	...	D3	PERSONAL	D3	333
FILOMENA	RASCAS	234	...	D1	ADMINISTRACION	D1	111
GUMERSINDA	MIMOS	543	...	NULL	NULL	NULL	NULL
NULL	NULL	NULL	...	NULL	TRAINING	D4	NULL

A veces, queremos que se muestre algo distinto de NULL en las columnas con dicho valor. Para ello, usamos la función COALESCE(columna, valor).

```
SELECT nombre, apellido, COALESCE(dep, 'SIN DEPARTAMENTO')  
FROM empleado;
```

La columna y el valor deben de ser del mismo tipo. Si no, podemos llevar a cabo transformaciones.

```
SELECT nss, nombre, COALESCE(TO_CHAR(nssjefe), 'SIN JEFE')  
FROM empleado;
```



↳ nssjefe es de tipo NUMBER

TABLAS COMO CONJUNTOS

Se les pueden aplicar operaciones de conjuntos a las tablas: UNION, INTERSECTION y MINUS.

- UNION: filas de ambas tablas (sin duplicados).
- INTERSECT: filas que estén en ambas tablas.

· MINUS: filas en la primera tabla que no estén en la segunda.

CONSULTAS ANIDADAS

Es una consulta SELECT completa, dentro de la cláusula WHERE de otra consulta.

Los valores que selecciona se usan en la condición de la consulta externa.

* Familiares de empleados que son directores de departamento

```
SELECT nombre FROM familiar  
WHERE nssemp IN (SELECT nssdire  
                  FROM departamento);
```

Primeros se evalúa la subconsulta,
luego se resuelve el WHERE, y
por último se resuelve el SELECT.

Es posible tener varios niveles de consultas anidadas.

```
SELECT dni, nombre  
      FROM empleado  
     WHERE dep IN (SELECT coddep  
                   FROM departamento  
                  WHERE nssdire IN (SELECT nss  
                                    FROM empleado  
                                   WHERE nombre = 'MACARENO'  
                                         AND apellido = 'SOSO'));
```

Si la consulta anidada devuelve una sola columna y fila y es un valor escalar,
se pueden usar los operadores (=, <, >, <=, >=, <>) en vez de IN.
Estos operadores se pueden concatenar con ANY, SOME o ALL.

* Nombres y apellidos de los empleados cuyo salario es menor que el de todos los empleados del departamento D3

```
SELECT nombre, apellido  
      FROM empleado  
     WHERE salario < ALL (SELECT salario  
                           FROM empleado  
                          WHERE dep = 'D3');
```

CORRELACIÓN

A veces una consulta anidada necesita un valor que está en un FROM exterior.

* Código de los departamentos cuyo director pertenece a otro departamento distinto
(detección de errores en los datos introducidos)

```
SELECT coddep  
      FROM departamento  
     WHERE nssdire IN (SELECT nss  
                           FROM empleado  
                          WHERE dep <> coddep);
```

Se dice que las consultas están correlacionadas.

OJO! Es una operación muy costosa.

Operador EXISTS: sirve para comprobar si una tabla es vacía (devuelve TRUE o FALSE).

Se usa sobre todo en sentido negativo (con NOT).

* Nombres y apellidos de los empleados sin familiares

```
SELECT nombre, apellido  
      FROM empleado  
     WHERE NOT EXISTS (SELECT *  
                           FROM familiar  
                          WHERE nssemp=nss);
```

FUNCIONES DE AGREGADOS

Es muy habitual necesitar hacer cálculos con los datos de una tabla.

Para ello podemos usar funciones como: `SUM()`, `MIN()`, `MAX()`, `AVG()`. Dentro se escribe el nombre de la columna con la que queremos operar.

Función `COUNT(*)` cuenta el nº de filas o, si en vez de * escribimos una columna, cuenta valores no nulos de ella.

Si queremos que cuente valores distintos: `COUNT(DISTINCT columna)`.

¿Cuántos empleados son jefes de algún otro?

```
SELECT COUNT(DISTINCT nssjefe)
FROM empleado;
```

* Apellidos, nombres y salarios de empleados cuyo sueldo coincide con el sueldo medio
SELECT nombre, apellido, salario
FROM empleado
WHERE salario = (SELECT AVG(salario)
FROM empleado);

apellido	nombre
vacio	

Las funciones de agregados...

- Se pueden anidar: `MAX(COUNT(*))`
- Admiten el DISTINCT (no solo el COUNT)
- SUM, MAX, MIN, AVG ignoran el NULL, pero lo pueden devolver si no hay filas.
- COUNT no ignora NULL, pero siempre devolverá un nº o un 0.

AGRUPACIÓN

A veces necesitamos agrupar valores para realizar algunos cálculos.

Esto se hace con la cláusula `GROUP BY`, que selecciona una columna por la que se agruparán los valores en la tabla.

columna de agrupación ↓

```
SELECT dep, COUNT(*), AVG(salario)
FROM empleado
GROUP BY dep;
```

↓
en el SELECT
solo podrá'
aparecer la
columna de
agrupación

dep	COUNT(*)	AVG(*)
D1	3	1100
D2	1	2100
D3	2	910
NULL	1	850

y funciones de agregadas

nombre	apellido	salario	...	dep
JONÁS	SOLANO	1100	...	D1
MACARENO	SOSO	1100	...	D1
FILOMENA	RASCAS	1100	...	D1
EUSEBIO	MULETAS	2100	...	D2
RIGOBERTA	CALAVERA	900	...	D3
CASIANA	FABERGÉ	920	...	D3
GUMERSINDA	MIMOS	850	...	NULL

Grupo 1: D1

Grupo 2: D2

Grupo 3: D3

Grupo 4: NULL

Usamos la cláusula `HAVING` para poner una condición a los grupos de filas. Es como un `WHERE` para grupos.

```
SELECT nombre, nssdire
FROM departamento
WHERE coddep IN (SELECT dep
                  FROM empleado
                  GROUP BY dep
                  HAVING AVG(salario) < 1200);
```

ONLINE VIEW

Es una consulta dentro de la cláusula FROM de otra SELECT. No es una anidada.

* Para cada empleado, mostrar su nombre, salario y cuántos familiares tiene.

Columnas: (nombre, salario, cuantos_familiares)

```
SELECT nombre, salario, n_familiares  
FROM empleado e
```

```
JOIN (SELECT nssemp, COUNT(*) n_familiares  
      FROM familiar  
      GROUP BY nssemp) f
```

ON e.nss = f.nssemp;

2. El JOIN obtiene una fila para cada empleado, junto con el nssemp y su número de familiares

1. La online view obtiene el número de familiares que tiene cada empleado

NOMBRE	SALARIO	N_FAMILIARES
MACARENO	1100	2
JONAS	1100	1
EUSEBIO	2100	1
RIGOBERTA	900	3

Se pueden hacer tantas Online Views como se quieran, siempre añadiendo luego la condición de reunión en el ON.

(en el power point hay más ejemplos)

Sirve para tener disponible otra tabla creada por nosotros para sacar columnas e información de ella

DIVISION EN SQL

Podemos encontrar un tipo de consultas que requieren la obtención de filas de una tabla que estén relacionadas con todas las de otra tabla.

Ejemplos:

"Clientes que tienen cuenta en todas las sucursales".

"Clientes que han hecho pedido de todos los productos"

Para resolverlo, se pone el enunciado de la consulta en negativo y luego expresarlo en SQL.

"Clientes para los cuales no hay un producto que no hayan pedido"

Consulta: Códigos de los clientes con cuenta en todas las sucursales que están en 'MURCIA'

▫ Código de los clientes tales que NO EXISTE una sucursal de Murcia en la que NO tengan cuenta

-- Dos correlaciones

```
SELECT codigo  
FROM cliente C  
WHERE NOT EXISTS  
(SELECT *  
  FROM sucursal S  
 WHERE ciudad = 'MURCIA'  
   AND NOT EXISTS  
    (SELECT *  
     FROM cuenta Q  
 WHERE Q.cliente_cod = C.codigo  
   AND Q.sucursal_cod= S.codigo));
```

2. Esta SELECT saca las cuentas del cliente C en la sucursal S

Soluciones

-- Una correlación (un poquito mejor que la anterior)

```
SELECT codigo  
FROM cliente C  
WHERE NOT EXISTS  
(SELECT *  
  FROM sucursal S  
 WHERE ciudad = 'MURCIA'  
   AND S.codigo NOT IN  
    (SELECT sucursal_cod  
     FROM cuenta Q  
 WHERE Q.cliente_cod = C.codigo));
```

La comparación que se hace en NOT IN evita la segunda comparación dentro de la subconsulta

EVALUACION

Sólo son obligatorios el SELECT y el FROM.

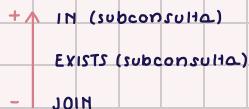
Orden de ESCRITURA
SELECT lista columnas o funciones
FROM lista tablas
WHERE condición para filas
GROUP BY lista columnas agrupación
HAVING condición para grupos
ORDER BY lista columnas ordenación

↓ Orden de EVALUACIÓN	
1. FROM lista tablas	Acceso a tablas, o reunión
2. WHERE condición para filas	Descarta las filas que incumplen la condición
3. GROUP BY lista columnas agrupación	Crea grupos de filas
4. HAVING condición para grupos	Descarta los grupos de filas que la incumplen
5. SELECT lista columnas y/o funciones	Para el conjunto de filas resultado, elige columnas indicadas, calcula operaciones y funciones
6. ORDER BY lista columnas ordenación	Ordena las filas del resultado

SQL ofrece flexibilidad, ya que una misma consulta se puede resolver de varias maneras.

Buenas prácticas

Se recomienda usar, en orden de elegancia y claridad:



Hemos de evitar el uso de SELECT * por temas de eficiencia (el SGBD accederá a metadatos)

Es bueno usar pseudónimos para las columnas cuando se realiza un JOIN

Usar sangrías y espaciados en la redacción de consultas.

Debemos evitar la correlación cuando sea posible. Deterioran mucho la eficiencia. Se puede traducir a una consulta usando JOIN

NO USAR el anti-JOIN →

es incorrecto (usar

NOT IN o WHERE).

SELECT e.nombre, e.apellido
FROM empleado e
JOIN familiar f
ON e.nss <> f.nssempp;

ANTI-JOIN
DETECTED

¡¡Aaaargh!!

En cuanto eficiencia:

+ ↑ IN sin correlación

JOIN

- IN y EXISTS con correlación

e.nombre	e.apellido	e.nss	e.sememp	numero	nombre	parentesco
MACARENO	SOSO	111 ...	123	1	JONAS	HIJO
MACARENO	SOSO	111 ...	321	1	RENATA	HIJO
MACARENO	SOSO	111 ...	321	1	RENYATA	HIJA
MACARENO	SOSO	111 ...	321	2	RÓMULIA	ABUELA
MACARENO	SOSO	111 ...	321	3	TORCUATA	ABUELA
JONAS	SOLANO	123 ...	222	1	ELEUTERIO	HIJO
JONAS	SOLANO	123 ...	222	1	SINFOROSA	ABUELA
JONAS	SOLANO	123 ...	321	1	RENATA	HIJA
JONAS	SOLANO	123 ...	321	2	RÓMULIA	ABUELA
JONAS	SOLANO	123 ...	321	3	TORCUATA	ABUELA
EUSEBIO	MULETAS	222 ...	123	1	JONAS	HIJO
EUSEBIO	MULETAS	222 ...	111	1	JULIANA	MADRE
EUSEBIO	MULETAS	222 ...	111	2	SINFOROSA	ABUELA
EUSEBIO	MULETAS	222 ...	321	1	RENATA	HIJA
EUSEBIO	MULETAS	222 ...	321	2	RÓMULIA	ABUELA
EUSEBIO	MULETAS	222 ...	321	3	TORCUATA	ABUELA
RIGOBERTA	CALAVERA	321 ...	123	1	JONAS	HIJO
RIGOBERTA	CALAVERA	321 ...	111	1	JULIANA	MADRE
RIGOBERTA	CALAVERA	321 ...	111	2	SINFOROSA	ABUELA
RIGOBERTA	CALAVERA	321 ...	222	1	ELEUTERIO	HIJO
FILOMENA	RASCAS	234 ...	123	1	JONAS	HIJO
FILOMENA	RASCAS	234 ...	111	2	JULIANA	MADRE

se combina cada empleado
con TODOS los familiares
que no sean suyos

(Aunque a veces un JOIN es
más eficiente)

Modificación de datos

Sentencias SQL que permiten introducir nuevas filas o modificarlas.

INSERT

Añade una fila completa a una tabla, incluye el nombre de la tabla y la lista de valores

INSERT INTO empleado

VALUES ('CEFERINA', 'SALSIPUEDES', 444, '44C',
TO_DATE('30-DIC-92', 'dd-MON-yy'),
'YECLA', 'S', 3700, 'D4', NULL, 0);

Aquí, los valores se dan en el orden en el que están las columnas en la tabla.

Si queremos dar los valores en otro orden:

orden específico

INSERT INTO empleado (nombre, apellido, nss, dni, dep, salario, nssjefe,
ciudad, fechanacim, est_civil, cuantos_familiares)
VALUES ('CEFERINA', 'SALSIPUEDES', 444, '44C', 'D4', 3700, NULL,
'YECLA', TO_DATE('30-DIC-1992', 'dd-MON-yyyy'), 'S', 0);

Es posible omitar valores de las columnas si estas aceptan NULL o si tienen valor por defecto.

El SGBD rechaza o acepta las inserciones según incumplan o no las restricciones asociadas a la tabla.

Es posible rellenar una tabla extrayendo su contenido de la base de datos, mediante una **SELECT**.

* Sea una tabla INFO_DEP vacía, creada mediante esta sentencia
CREATE TABLE info_dep (
nombre_dep VARCHAR2(25) NOT NULL,
num_empleados NUMBER(3) DEFAULT 0 NOT NULL,
suma_salarios NUMBER(8) NOT NULL
);

► Almacenará el nombre de cada departamento, cuántos empleados tiene y la suma de los salarios de sus empleados

INSERT INTO info_dep (nombre_dep, num_empleados, suma_salarios)
SELECT d.nombre, COUNT(*), SUM(salario)
FROM departamento d JOIN empleado e ON d.coddep = e.dep
GROUP BY d.nombre;

OJO! info_dep no será una tabla actualizada.

* Crear una tabla para almacenar datos de empleados nacidos antes de 1960

CREATE TABLE emp_veterano (nss, nombre, jefe, depto)
AS SELECT nss, nombre || '-' || apellido, nssjefe, dep
FROM empleado
WHERE EXTRACT(YEAR FROM fechanacim) < 1960;

EMP_VETERANO			
nss	nombre	jefe	depto
111	MACARENO SOSO	NULL	D1
123	JONÁS SOLANO	111	D1
333	CASIANA FABERGÉ	123	D3

Otra manera de crear

tablas usando:

CREATE TABLE <tabla> AS SELECT ..

DELETE

Elimina filas completas de una tabla (solo una en el FROM):

```
DELETE FROM empleado  
WHERE apellido = 'ROLLER';  
DELETE FROM departamento  
WHERE nssdire = 111;  
DELETE FROM empleado  
WHERE dep IN  
(SELECT coddep  
FROM departamento  
WHERE nombre = 'INVESTIGACION');
```

- 1º se accede a la tabla
- 2º se ejecuta el WHERE
- 3º se ejecuta el borrado

El WHERE servirá para seleccionar las filas que se quieran borrar.

Si no hay WHERE, se eliminarán todas las filas. !!

UPDATE

Modifica valores de columnas de una o más filas de una tabla.

* Cambiar el nombre del departamento D4 por 'FORMACION' y asignarle un director

```
UPDATE departamento  
SET nombre = 'FORMACION', nssdire = 444  
WHERE coddep = 'D4';  
  
UPDATE empleado SET nssjefe = NULL  
WHERE apellido = 'FUERTES';  
UPDATE empleado SET salario = DEFAULT  
WHERE nss = 333;
```

- 1º se accede a la tabla
- 2º se ejecuta el WHERE
- 3º se ejecuta el SET

SET → modifica (valor, NULL, DEFAULT)

WHERE → selecciona

→ Ejemplos ←

* Asignar a todo director de departamento un salario igual a la media de los salarios de su departamento más el 25% de dicha media

UPDATE empleado d SET salario = (SELECT AVG(salario) * 1.25

FROM empleado e
WHERE e.dep = d.dep)

1º selecciona sólo las filas de EMPLEADO que son directores

WHERE nss IN (SELECT nssdire
FROM departamento);

2º para cada una de esas filas, calcula la media de su departamento y le asigna el nuevo salario

Tema 9: Reglas de integridad y otros

Reglas de integridad

Al crear un esquema de BD Relacional, estas restricciones quedan almacenadas junto con los datos. Formarán parte de las **metadatos**.

La **integridad** es la consistencia o corrección de los datos almacenados en la base de datos.

COMPONENTES

- **Nombre**: aparece en los mensajes de error.
- **Restricción**: expresión booleana
- **Respuesta a un intento de incumplimiento**

□ Nombre departamento_director_ok
Restricción NOT EXISTS (SELECT *

Todo director de
departamento debe ser
uno de sus empleados

Respuesta a un intento de incumplimiento
Siempre RECHAZAR

Un INSERT, un UPDATE o un
DELETE podría incumplir
esta 3^a regla

FROM departamento
WHERE nssdire NOT IN (SELECT nss
FROM empleado
WHERE dep = coddep)

ejemplos

□ Nombre emp_fk_emp -- en EMPLEADO
Restricción FOREIGN KEY(nssjefe) REFERENCES EMPLEADO(nss)
Los valores de "nssjefe" deben existir en la columna "nss" de EMPLEADO
Respuesta a un intento de incumplimiento
En el INSERT: RECHAZAR; En el DELETE: SET NULL; En el UPDATE: CASCADE

CATEGORÍAS

1. De dominio (no las veremos)
2. De tabla
3. Generales

REGLAS DE INTEGRIDAD DE TABLA

Restricción asociada específicamente a una tabla. No existe si la tabla no existe.

Se especifica dentro de CREATE TABLE. Son comprobadas inmediatamente.

Pueden ser:

- Restricciones de columna
- Claves: primaria, alternativa o ajena.
- RI añadida con ALTER TABLE.

REGLAS DE INTEGRIDAD GENERALES

Se refieren a los **assertos**. Son restricciones que van más allá de dominio y tabla.

[BD de ventas de productos] Un cliente puede conseguir un 15% de descuento en sus compras siempre que sea cliente desde hace más de 3 años y haya comprado más de 2 productos al año. ← ejemplo

[BD de una universidad] Un alumno puede matricularse de un máximo de 9 créditos de libre elección.

Incluyen un predicado con la condición que se debe cumplir y puede involucrar varias columnas y tablas.

```
CREATE ASSERTION RI_libro_tiene_copias  
CHECK (NOT EXISTS (SELECT *  
                   FROM libro  
                   WHERE isbn NOT IN (SELECT isbn  
                           FROM ejemplar)));
```

} nombre
} condición → en negativo

Son un elemento independiente de tablas y vistas.

Para eliminar un aserto: `DROP ASSERTION <nombre_aserto>`

Oracle no implementa la creación de asertos, pero podemos ejecutar la `SELECT` de la condición y si se cumple el aserto, nos devolverá una tabla vacía. !!

*Todo departamento debe tener al menos un empleado

► No existe departamento cuyo código no esté referenciado desde un empleado

```
CREATE ASSERTION RI4_empleado_en_departamento  
CHECK (NOT EXISTS (SELECT * FROM departamento  
                   WHERE coddep NOT IN (SELECT dep  
                           FROM empleado)));
```

Existen unas reglas llamadas pseudo-reglas de integridad que son:

- Especificación del tipo de datos
- Definición de vista (más adelante)

↳ vienen implícitamente

COMPROBACIÓN DE RESTRICCIONES

```
UPDATE empleado SET dep = 'D3' WHERE nss = 543; -- Gumersinda Mimos
```

► Comprobación de reglas de integridad:

- Tipo de datos de dep: el valor indicado es compatible
- Integridad referencial (clave ajena):
 - existe una fila en `DEPARTAMENTO` con `coddep = 'D3'`
- Aserto RI1a (entra al departamento D3: su salario debe ser $\geq 900\text{€}$)
- Aserto RI5_... (salario inferior al del director de su nuevo departamento)

Cuando existe un ciclo referencial (una tabla referencia a otra que referencia a la primera):

```
ALTER TABLE empleado  
  DISABLE CONSTRAINT emp_fk_dep;  
INSERT INTO empleado(...)  
VALUES (...);  
...  
INSERT INTO empleado(...)  
VALUES (...);  
INSERT INTO departamento(...)  
VALUES (...);  
...  
INSERT INTO departamento(...)  
VALUES (...);  
ALTER TABLE empleado  
  ENABLE CONSTRAINT emp_fk_dep;
```

Desactivar una de las claves ajenas:
es necesario saber su nombre

Insertar filas en la tabla en la que se ha desactivado la clave ajena: las referencias a la otra tabla NO se comprueban (se han desactivado)

Insertar filas en la otra tabla: las referencias hacia la 1^a tabla si se comprueban (no se han desactivado)

Reactivar la restricción de clave ajena: en este momento se comprueban las referencias (que se habían desactivado)

Vistas relacionales

Es una tabla obtenida como resultado de una consulta. Permite tratar a ese resultado

CREATE VIEW personal

AS SELECT nss, nombre, nssjefe, dep
FROM empleado;

como a una tabla

Está siempre actualizada, porque se crea cada vez que se consulta.

Una vista en sí no contiene información, si no que la deja ver

Se usan para almacenar consultas complejas, presentar los datos con una perspectiva diferente, ocultar la complejidad de los datos, proporcionar seguridad, aislar a las aplicaciones de los cambios en la definición de las tablas base.

Podemos definir nuevos nombres para las columnas. Esto es obligatorio cuando alguna

CREATE VIEW familiar_empleado (**empleado, familiar, parentesco**)
AS SELECT e.nombre, f.nombre, f.parentesco
FROM empleado e **JOIN** familiar f **ON** e.nss = f.nssemp;

columna es una concatenación o cálculo.

Es posible generar nuevas columnas en la vista, aplicando operaciones sobre las columnas extraídas.

Además, una vista puede aparecer en el **FROM** para crear otra vista.

CREATE VIEW directivo (nss, nombre, coddep, departamento)
AS SELECT p.nss, p.nombre, d.coddep, d.nombre
FROM personal p **JOIN** departamento d
 ON p.dep = d.coddep
WHERE p.nss **IN** (**SELECT** nssjefe
 FROM empleado);

PERSONAL es una vista creada anteriormente

Vemos que una vista puede ser "tabla base" de otra vista

① **CREATE VIEW** personal
AS SELECT nss, nombre, nssjefe, dep
FROM empleado;

VISTAS EN ORACLE

Creación:

CREATE [**OR REPLACE**] [[**NO**] **FORCE**] **VIEW** *nombre_vista*
[(*lista_nombres_columnas*)]
AS *consulta_definición*
[**WITH** { **READ ONLY**
| **CHECK OPTION** } [**CONSTRAINT** *nombre_restricción*]]

① La **CHECK OPTION** la veremos después

OR REPLACE: reemplazar si ya existe

FORCE: crear la vista aunque no existan las tablas base o no haya permisos.

WITH READ ONLY: prohíbe hacer **UPDATE**, **INSERT** y **DELETE** sobre las tablas base a través de la vista

Si lo que se desea es modificar la vista usamos la opción CREATE OR REPLACE.

Se pueden hacer consultas con vistas como si fueran tablas.

Sentencia de usuario	Traducción (lo que realmente se ejecuta)
SELECT * FROM veterano WHERE dep = 'D1';	SELECT nombre, dni, nss, fechanacim, dep FROM empleado WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy') AND dep = 'D1';

```
CREATE OR REPLACE  
VIEW veterano  
AS SELECT nombre, dni, nss,  
fechanacim, dep  
FROM empleado  
WHERE fechanacim <  
TO_DATE('01/01/1970',  
'dd/mm/yyyy');
```

También se pueden manipular datos a través de estas

Sentencia de usuario	Traducción (lo que realmente se ejecuta)
INSERT INTO veterano (nombre, apellido, nss, dni, fechanacim, salario, dep) VALUES ('EVA','EME',789,'78E', TO_DATE('09/07/1967', 'dd/mm/yyyy'), 980,'D4');	INSERT INTO empleado (nombre, apellido, nss, dni, fechanacim, salario, dep) VALUES ('EVA','EME', 789, '78E', TO_DATE('09/07/1967', 'dd/mm/yyyy') 980, 'D4');

Por otro lado, actualizar a través de una vista definida sobre varias tablas base puede dar lugar a ambigüedad (que puede traducirse a 2 o más actualizaciones distintas). Ejemplo detallado en el ppt.

En general, podemos afirmar que:

- Una vista con una sola tabla base (teniendo una columna clave primaria o alternativa) sí es actualizable.
- Una vista definida sobre varias tablas con JOIN, no lo es.
- Una vista definida con agrupación y funciones de agregados no lo es

Si añadimos WITH CHECK OPTION a la creación de vistas, el SGBD comprobará cada INSERT y UPDATE que se haga sobre la vista.

```
CREATE VIEW precario  
AS SELECT nombre, apellido, nss, dni, salario, dep  
FROM empleado  
WHERE salario < 550  
WITH CHECK OPTION;
```

Así, tanto el INSERT como el UPDATE anteriores fallarían

Se comprueba que se cumple la definición de la vista.

En Oracle se le puede además dar nombre a la cláusula, para que aparezca en los mensajes de error.

```
CREATE OR REPLACE NOFORCE VIEW plantilla  
AS SELECT nombre, apellido, nss, dni, salario, dep  
FROM empleado  
WHERE salario < 3000 -- se ocultan datos de "jefazos y jefazas"  
WITH CHECK OPTION CONSTRAINT plantilla_const;
```

Las vistas son un elemento **virtual**, no ocupa espacio de almacenamiento. Solo se almacena su definición (en el INFORMATION_SCHEMA).

Las vistas se **eliminan** con: `DROP VIEW vista;`

Si formaban parte de la base de otra vista, esa no se elimina pero se marca como inválida.

índices

Son un concepto que pertenece al **nivel físico**.

Vamos a asumir que cada tabla se guarda en un fichero. Cada fila en un registro.

Los ficheros ocupan bloques y páginas de memoria.

¿Cómo encontramos una fila o varias en una tabla?

Una opción es la **búsqueda secuencial**. Pero es muy ineficiente y requiere muchas lecturas de bloque.

Fichero EMPLEADO

nss	nombre	...	fechanacim	dep
123	JONÁS		10/10/1945	D1
321	RIGOBERTA		12/11/1974	D3
222	EUSEBIO		01/01/1969	D2
111	MACARENO		06/04/1944	D1
333	CASIANA		15/06/1943	D3
234	FILOMENA		18/07/1970	D1
543	GUMERSINDA		10/02/1980	NULL
444	CEFERINA		30/12/1992	D4
456	PRUDENCIO		22/01/1995	D4

Fichero con 5 bloques
Cada bloque contiene 2 registros

Un índice es una estructura de datos auxiliar que permite acelerar el acceso a las filas de una tabla con base en los valores de una o más columnas.

En la tabla empleado, la columna de **indexación** podría ser el nss.

Índice
IDX_NSS_EMPLEADO

nss	bloque
111	B2
123	B1
222	B2
234	B3
321	B1
333	B3
444	B4
456	B5
543	B4

1º Coger todos los valores del campo de indexación
2º Ordenarlos
3º Para cada uno de los valores, crear una entrada de índice.
El campo "nss" contendrá el valor.
El campo "bloque" contendrá un puntero al bloque de EMPLEADO en el que está el registro con tal valor en "nss"

Fichero EMPLEADO

bloque	nss	nombre	...
B1	123	JONÁS	
B2	222	EUSEBIO	
B3	333	CASIANA	
B4	543	GUMERSINDA	
B5	444	CEFERINA	
	456	PRUDENCIO	

↓
puntero al bloque

Se pueden crear varios índices sobre una misma tabla. Este ocupa poco.

Oracle crea índices automáticos con clave primaria y alternativas.

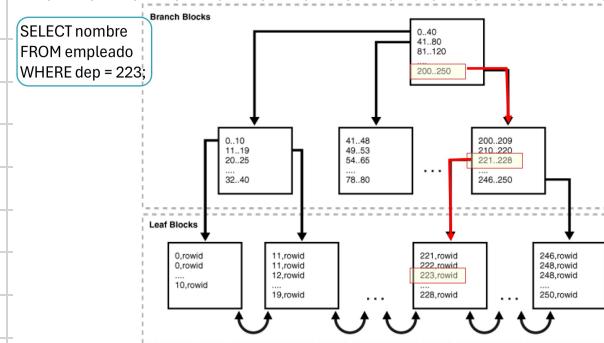
A veces crea índices con columnas no clave que se usan mucho en el WHERE y JOIN pero sólo si:

- La tabla tiene muchas datos
- Sus consultas más frecuentes recuperan un bajo porcentaje de datos (menos del 20%).

```
CREATE INDEX nombre_idx
ON nombre_tabla(columna [ASC | DESC]
[, columna [ASC | DESC]...])
...;
```

Son mantenidas por el SGBD, son independientes de los datos de la tabla.

Los más comunes son los índices en árbol B.



¿Qué es el rowid?

Identificador de fila. Contiene el fichero, el bloque y la posición, codificado en base 64.

AAAPzSAAEAAAABzAAA

Para eliminar un índice: `DROP INDEX nombre;`

se hace cuando la tabla es pequeña o hay pocas entradas en el índice, o cuando no se esperan hacer más consultas con esa columna de indexación.

Si queremos eliminar un índice asociado a una clave hay que desactivar o eliminar la restricción de clave primaria.

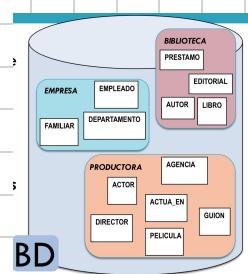
Esquema

Una base de datos puede contener varios esquemas.

Un esquema agrupa tablas y otros elementos.

```
CREATE SCHEMA nombre_de_esquema
AUTHORIZATION identificador_de_autorización
    ~usuario
```

Un usuario puede crear una tabla en el esquema de otro si tiene permisos para ello.



En Oracle:

```
CREATE SCHEMA AUTHORIZATION cuenta;
```

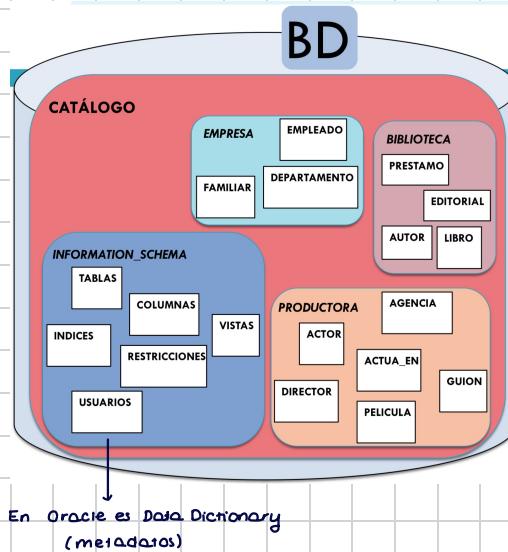
No hace literalmente nada (sólo un esquema por cuenta y no tiene nombre).

Tampoco permite eliminar esquemas, que en otro lenguaje sería:

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

↓ ↓
sólo lo elimina elimina todo
si no tiene nada

Un **catalogo** recoge todos los esquemas de un mismo entorno SQL.



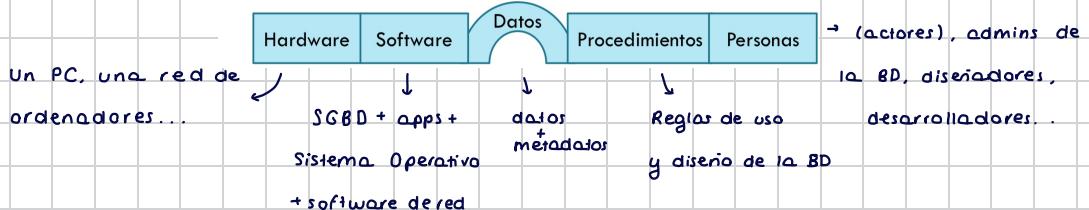
Puede haber claves ajena que refieren a tablas de distintos esquemas pero mismo catálogo.

Para acabar, recordemos.

BD = DATOS + METADATOS

Tema 10: Sistemas de bases de datos

Componentes



ACTORES

- Administrador de la base de datos (ABD):

Administra la BD, el SGBD y el software de aplicaciones o programas de acceso. Implementa la estructura de la BD y restricciones de los datos. Crea / modifica estructuras de almacenamiento. Controla la seguridad (permisos). Define las copias de seguridad. Garantiza el funcionamiento correcto del sistema.

- Diseñadores de la base de datos:

Recogen las necesidades y requisitos del cliente. Con ello crean **esquemas conceptuales (E-R)** identificando relaciones. Eligen las estructuras para representar los datos.

- Desarrolladores de Software:

- Analistas de sistemas: determinan necesidades de procedimiento y especifican las operaciones.
- Desarrolladores de aplicaciones: implementan las especificaciones.

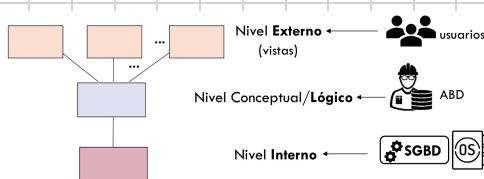
- Usuarios finales:

Son los clientes.

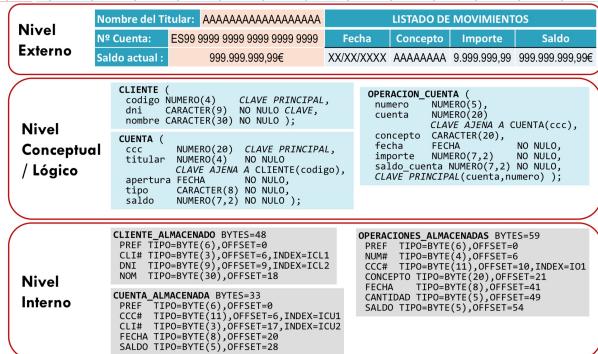
- Inexpertos: no son conscientes de la existencia del SGBD y acceden a la BD mediante sencillos programas de aplicación.
- Avanzados: conocen el SGBD. Pueden usar SQL.

Arquitectura de 3 niveles ANSI - SPARC

La complejidad de un sistema de bases de datos se oculta mediante niveles de abstracción.

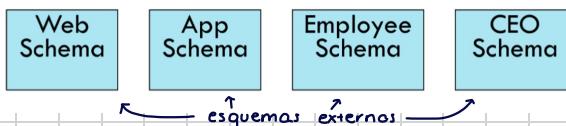


Con esto logramos aislar los distintos niveles y hacerlos independientes: un cambio en un aspecto físico de almacenamiento no debería afectar a la estructura de la BD.



NIVEL EXTERNO O DE VISTAS

Describe que parte de los datos es relevante para cada usuario o aplicación.



Para cada tipo de usuario se le da un esquema.

Pueden existir varias vistas del mismo esquema.

NIVEL LÓGICO

Describe qué datos están almacenados en la BD y sus relaciones. Usa el esquema conceptual o lógico.

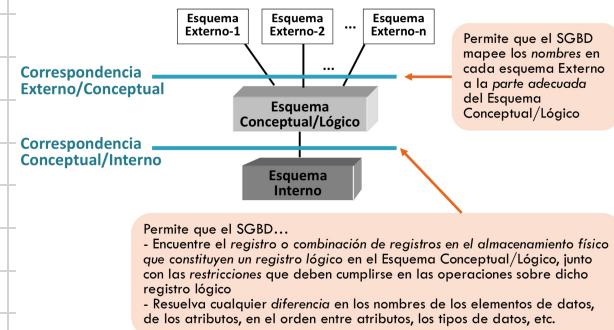
- Esquema conceptual → más cercano al usuario
- Esquema lógico → más cercano al SGBD

NIVEL INTERNO

Describe cómo los datos están almacenados en la BD. Usa el esquema interno: asigna el espacio para datos e índices, describe los tipos de registros almacenados, define estructuras de acceso... Muy cercano al nivel físico pero no trata con registros físicos.

Los 3 niveles de la A3N son descripciones de datos. Los datos reales están solo en el nivel físico.

El SGBD es el responsable de mantener la correspondencia entre niveles. Estas correspondencias están guardadas en el INFORMATION_SCHEMA (metadatos). Se debe comprobar la consistencia entre ellas.



Independencia de datos

"Capacidad de modificar el esquema de un nivel sin tener que cambiar el esquema del nivel inmediato superior".

INDEPENDENCIA LÓGICA

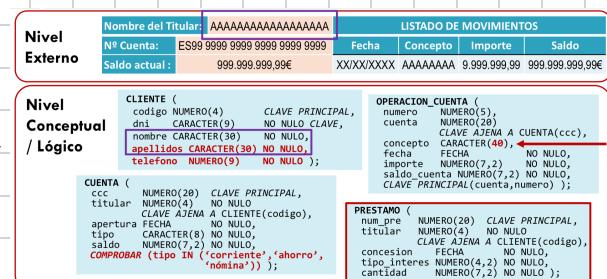
Modificar el esquema lógico/conceptual sin alterar esquemas externos.



El esquema externo permanece inalterado.

So lo hay que modificar la correspondencia entre esquemas:

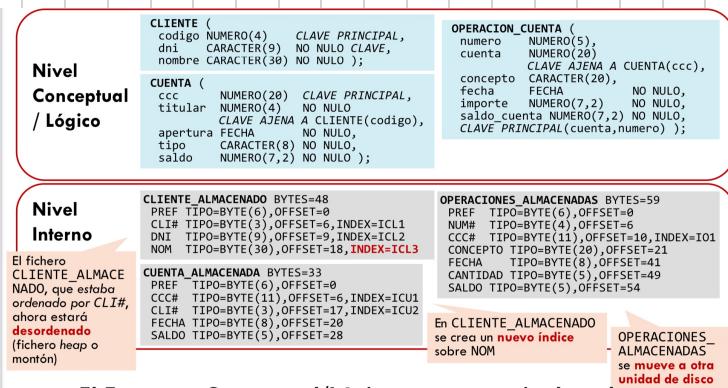
"Nombre del titular" ahora concatena "nombre" y "apellidos".



INDEPENDENCIA FÍSICA

Capacidad de modificar el esquema interno sin alterar el esquema lógico/conceptual.

Lo único que los usuarios notarán es una mejora en el rendimiento.



En resumen: gracias a la arquitectura de 3 niveles y a los metadatos; la modificación de un nivel no modifica un nivel superior, si no la correspondencia entre niveles.

El único inconveniente es que mantener las correspondencias actualizadas supone un coste en eficiencia.

Además, la independencia de datos incluye la independencia entre programas y datos.

Funciones del SGBD

1 ALMACENAMIENTO, RECUPERACIÓN Y ACTUALIZACIÓN DE DATOS

(tema 11)

2 SERVICIO PARA PERMITIR INDEPENDENCIA DE DATOS

A3N ✓

3 METADATOS ACCESIBLES PARA EL USUARIO

INFORMATION_SCHEMA ✓

4 SERVICIOS DE INTEGRIDAD ✓

5 SOPORTE DE TRANSACCIONES

Transacción: secuencia de acciones que lee y/o actualiza el contenido de la BD.

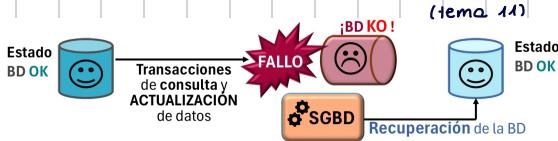
Soporlte: asegura que, o todos los cambios se han hecho correctamente, o que no se ha hecho ninguno de los cambios.

Se garantiza incluso en presencia de fallos y teniendo en cuenta la concurrentia de varios usuarios.

6 SERVICIO DE CONTROL DE CONCURRENTIA

Garantiza que la BD es actualizada correctamente cuando múltiples usuarios/aplicaciones modifican los datos a la vez

7 SERVICIO DE RECUPERACIÓN DE FALLOS



Recuperar las BD tras fallos:
system crash, errores hardware,
errores software...

8 SERVICIO DE AUTORIZACIÓN (SEGURIDAD)

Control de acceso selectivo

1. Solo usuarios autorizados. → cuentas de usuario y contraseña
2. Solo a ciertas partes de la BD.] → restricciones para cada cuenta
3. Solo para realizar ciertas operaciones.

9 SOPORTE PARA COMUNICACIÓN DE DATOS

Garantiza que los usuarios accedan a una BD centralizada desde ubicaciones remotas.



10 UTILIDADES ADICIONALES

Herramientas de importación/exportación de datos, utilidades de monitorización y supervisión, análisis estadístico, herramientas de backup, etc.

Tema II: Procesamiento de transacciones

Soporte de transacciones

Una transacción es una acción, o una secuencia de acciones, que lee y/o actualiza el contenido de la base de datos. Se ejecuta como una unidad: es una unidad lógica de procesamiento.

Ejemplo. transferencia bancaria

Transferir 200€ de una cuenta X a otra cuenta Y

1. Leer el saldo de X
2. Comprobar que el saldo de X es superior a 200
Si no, indicar que no se puede realizar la transferencia y finalizar
3. Restar 200 del saldo de X
4. Sumar 200 al saldo de Y
5. ...

Transferir 200€ de una cuenta X a otra cuenta Y

1. SELECT saldo
FROM cuenta
WHERE ccc = X;
2. SI (saldo<200) ENTONCES
ko_saldo_insuficiente(saldo);
3. UPDATE cuenta
SET saldo=saldo - 200
WHERE ccc = X;
4. UPDATE cuenta
SET saldo=saldo + 200
WHERE ccc = Y;
5. ...

¿Qué pasa si hay un error?

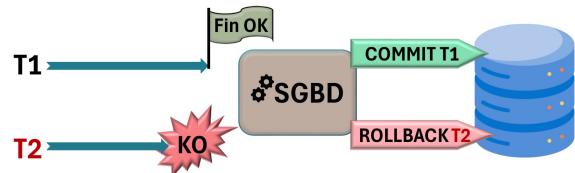
Una transacción es ATÓMICA, o se ejecuta completa, o no se ejecuta

SOPORTE DE TRANSACCIONES

Toda transacción termina con éxito o con fracaso.

ROLLBACK

↓ COMMIT



PROPIEDADES ^{"ACID"} DE UNA TRANSACCION

A Atomicidad

Responsable

Subsistema de Recuperación del SGBD

C Consistencia

Programadores +
Subsistema de Integridad del SGBD

I Isolation (aislamiento)

Subsistema de Conurrencia del SGBD

D Durabilidad

Subsistema de Recuperación del SGBD

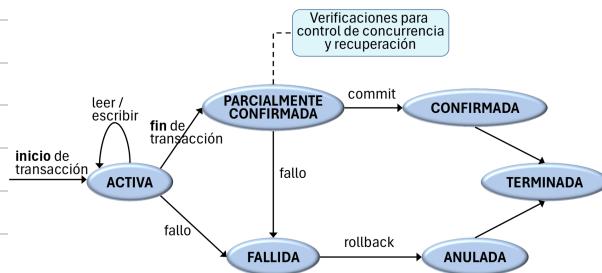
Atomicidad: ejecución "Todo o nada"

Consistencia: una transacción lleva la BD de un estado de consistencia a otro estado de consistencia.

Isolation (aislamiento): la ejecución de una transacción es independiente de las otras.

Durabilidad: los cambios se guardan en la BD permanente.

Estatos de una transacción:



Inicio de una transacción → al ejecutar una sentencia SQL incluida o incluida en un programa.

Finalización de una transacción → al ejecutar explícitamente COMMIT o ROLLBACK, al terminar el programa en ejecución.

SOPORTE DE TRANSACCIONES (EN ORACLE)

Una transacción inicia en Oracle cuando:

- No hay una transacción en proceso y se ejecuta una sentencia LMD: INSERT, UPDATE, DELETE, SELECT
- Cuando se ejecuta una sentencia LDD: CREATE, ALTER, DROP, RENAME, COMMENT
(si hay ya una transacción en proceso, Oracle la finaliza con COMMIT y comienza la nueva)

Se finaliza con:

- COMMIT explícito: ejecución de la sentencia COMMIT.
- COMMIT implícito: el programa finaliza sin errores. Cada sentencia LDD es tratada como una transacción.
- ROLLBACK explícito
- ROLLBACK implícito → errores, cerrar la ventana de la BD directamente

Conexión a SQL Developer	(inicio de sesión interactiva)
SELECT ...;	Inicia T1
SELECT ...;	
UPDATE...;	
DROP TABLE...;	Finaliza T1 (COMMIT) Inicia T2 y finaliza T2 (COMMIT)
INSERT INTO...;	Inicia T3
UPDATE...;	
SELECT...;	
INSERT INTO...;	
ROLLBACK;	Finaliza T3 (KO) Deshace TODA modificación de T3
SELECT ...;	Inicia T4
DELETE ...;	
INSERT INTO...;	
INSERT INTO...;	
COMMIT;	Finaliza T4 (COMMIT)
Desconexión del SQL Developer	