

T3: Implementación



1

Tema 3: Implementación BBDD

Lenguaje SQL

- Lenguaje declarativo
- Basada en álgebra
- cálculo relacional

Lenguajes que proporciona DBMS

- Lenguaje de Definición de Datos (DDL) ▶ CREATE, DROP, ALTER
Diseñar y crear esquemas fijos
- Lenguaje de Control de Datos (DCL) ▶ GRANT, REVOKE, COMMIT, ROLLBACK
Acceso
Instrucciones para definir mecanismos de seguridad
- Lenguaje de Manipulación de Datos (DML) ▶ INSERT, UPDATE, DELETE, SELECT
almacenar, borrar, recuperar, modificar

SQLite

- Multiplataforma
- Modo transaccional
- "Cero configuración" → sin autodenominamiento
- admin
- BBDD almacenado en un único fichero
- Soporte → BBDD grandes (terabytes)
- Proporciona cliente en modo texto

Abrir una base de datos existente

```
shell> sqlite3 ruta/chinook.db
```

```
sqlite> .open ruta/chinook.db
```

Crear una base de datos que no existe

```
shell> sqlite3 ej1.db
SQLITE version 3.15.0 2016-10-14 10:20:30
Enter ".help" for usage hints.
sqlite> create table tb1(name varchar(10), two smallint);
sqlite> insert into tb1 values('hello!', 10);
sqlite> insert into tb1 values('goodbye!', 20);
sqlite> select * from tb1;
hello!10
goodbye!20
sqlite>
```

```
sqlite> CREATE TABLE tb12 (
...>   f1 varchar(30) primary key,
...>   f2 text,
...>   f3 real
...> );
```

Guardar una base de datos en disco, si previamente no se había creado

```
sqlite> .save ruta/nombre.db
```

Listar las tablas de una base de datos

```
sqlite> .tables
```

Mostrar sentencias de creación de tablas e índices

```
sqlite> .schema
```

```
sqlite> .schema nombre_tabla
```

Listar todas las bases de datos abiertas en la actual conexión

```
sqlite> .databases
```

Escribir los resultados de consultas en un fichero

```
sqlite> .output nombre_fichero
```

Sugerencias para usar SQLite

④ BBDD de IoT

⑤ Formato de ficheros de aplicaciones

⑥ BBDD de sitios web

⑦ Análisis de datos

⑧ BBDD en el servidor

⑨ Demos y pruebas

⑩ Educación

⑪ ...

No recomendado para

④ APPS con arquitectura Cliente/Servidor

[Estados separados, APPS y son accedidos a través de la red]

⑤ Sitios web con un tráfico elevado

⑥ Conjunto datos muy grande (> 100 Terabyte)

⑦ Sist. con alta concurrencia (en escritura)

Tema 3: Implementación BBDD

Creación de BBDD con SQL

Lenguaje DDL proporciona:

- ① creación, modificación y eliminación → relaciones
- ② Consultar → esquema de cada relación (tablas)
- ③ Establecer: dominio de los valores → atributo (campo)
- ④ Establecer: restricciones Integridad
- ⑤ Gestionar conjunto índices asociado a cada relación
- ⑥ Establecer: estructura almacenamiento físico de cada relación (tabla) en disco

Tipos de datos en SQL

> Alfanuméricos:

- CHAR(n): Longitud fija
- VARCHAR(n): Longitud variable
- TEXT: ... más genérico

> Numéricos:

- DECIMAL(precisión,escala)
 - Precisión: antes de la coma
 - Escala: después de la coma
- INTEGER
- REAL

> Fechas y horas:

- TIMESTAMP: yyyy-mm-dd hh:mm:ss
- DATE: yyyy-mm-dd
- TIME: hh:mm:ss

> Otros tipos de datos:

- BLOB

→ SQLite: utiliza sust. de tipos dinámico → débilmente tipado

→ Define las siguientes clases de almacenamiento.

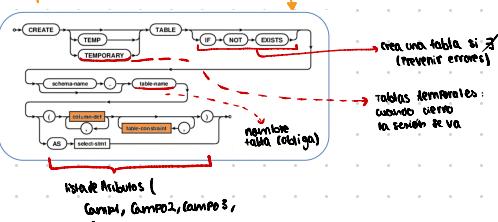
Tipo	Descripción
NULL	Valor nulo
INTEGER	Valor entero con signo, representado con 1, 2, 3, 4, 6 u 8 bytes
REAL	Valor real en coma flotante (8 bytes)
TEXT	Cadena de caracteres (string), utilizando codificación UTF-8
BLOB	Almacena valores binarios (de tamaño grande)

cada almacenamiento → se le asocia → clase de almacenamiento

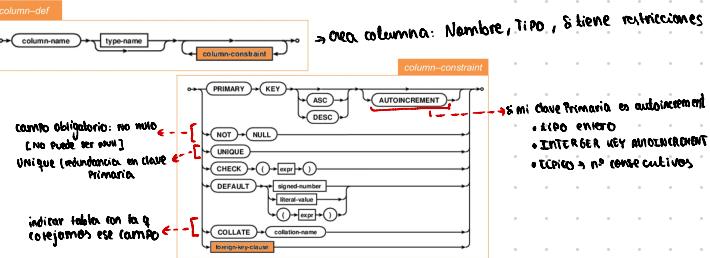
• booleanos → almacenan 0's y 1's
 • Fecha y Tiempo → almacenan TEXT → (YYYY-MM-DD HH:MM:SS.SS)

SQLite → "afinidad de datos" → tipo recomendado

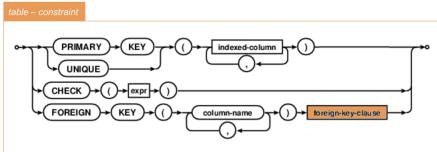
Creación de Relaciones (tablas)



Restricción sobre columnas



Restricciones sobre tablas

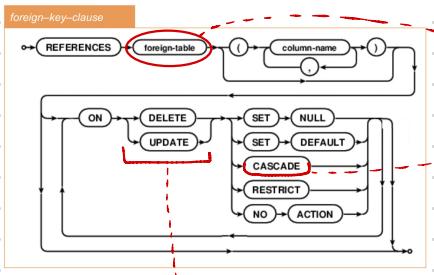


Ejemplos:

```
CREATE TABLE employees (
    employee_id INTEGER PRIMARY KEY AUTOINCREMENT,
    last_name VARCHAR NOT NULL,
    first_name VARCHAR,
    hire_date DATE
);

CREATE TABLE products (
    product_id INTEGER PRIMARY KEY AUTOINCREMENT,
    product_name VARCHAR NOT NULL,
    quantity INTEGER NOT NULL DEFAULT 0
);
```

Restricciones de Integridad Referencial (claves foráneas)



Cliente (idCliente...)
 Factura (idFactura, ..., idCliente)
 Que cliente está relacionado con qué factura

Replica en cascada: si idCliente = 5 → 6 ; 8 → 5
 → si idCliente → borrar → null/default, ...
 → si idCliente → actualizar → ...

Tema 3: Implementación BBDD

Ejemplos de Creación de Tablas:

```
CREATE TABLE departments (
    dep_id INTEGER NOT NULL PRIMARY KEY,
    name TEXT,
    location TEXT
);
```

```
CREATE TABLE employees (
    emp_id INTEGER NOT NULL,
    name TEXT UNIQUE,
    manager TEXT UNIQUE,
    salary REAL,
    bonus REAL,
    job TEXT,
    hiredate DATE,
    dep_id INTEGER,
    PRIMARY KEY (emp_id),
    FOREIGN KEY (dep_id) REFERENCES departments(dep_id)
        ON DELETE SET NULL ON UPDATE CASCADE
);
```



```
CREATE TABLE contacts (
    contact_id integer PRIMARY KEY,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    phone TEXT NOT NULL UNIQUE
);
```

```
CREATE TABLE groups (
    group_id integer PRIMARY KEY,
    name text NOT NULL
);
```

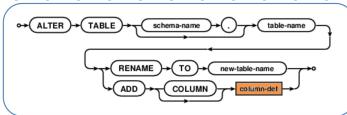
```
CREATE TABLE contact_groups (
    contact_id integer,
    group_id integer,
    PRIMARY KEY (contact_id, group_id),
    FOREIGN KEY (contact_id) REFERENCES contacts (contact_id)
        ON DELETE CASCADE ON UPDATE NO ACTION,
    FOREIGN KEY (group_id) REFERENCES groups (group_id)
        ON DELETE CASCADE ON UPDATE NO ACTION
);
```

```
CREATE TABLE company (
    com_id text(4) NOT NULL UNIQUE PRIMARY KEY,
    com_name text(15) NOT NULL
);
```

```
CREATE TABLE item(
    item_id text(4) NOT NULL UNIQUE PRIMARY KEY,
    item_desc text(20) NOT NULL,
    rate integer NOT NULL,
    item_code TEXT(4),
    FOREIGN KEY (item_id)REFERENCES company(com_id)
        ON UPDATE RESTRICT ON DELETE RESTRICT
);
```

Creación de BBDD con SQL

Modificación de Relaciones



Renombrar Tabla → `ALTER TABLE clientes RENAME TO ant_clientes;`

Añadir nueva columna → `ALTER TABLE ant_clientes ADD COLUMN sex char(1);`

Borrado de Relaciones



`DROP TABLE employees;` `DROP TABLE IF EXISTS orders;`

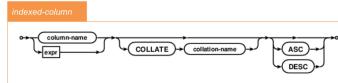
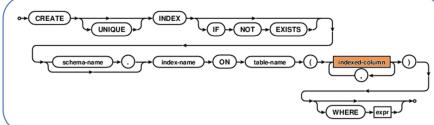
`DROP TABLE IF EXISTS miBaseDatos.facturas;`

Índices de Tablas

Índice ⇒ estructura de datos q reunite mejorar el rendimiento

- ① (elemento, posición)
- ② acelerar consultas
- ③ penaliza: actualizaciones, inserciones, borrados
- ④ 1 tabla → varios índices sobre 1 o más campos
- ⑤ se construyen utilizando < tablas, B+, B* tablas, hash >
- ⑥ necesitar q los campos sean únicos

Creación Índices



`CREATE INDEX idx_cli ON clientes (nombre);`

`CREATE INDEX idx_cli ON clientes (fecha DESC);`

`CREATE UNIQUE INDEX ix_emp ON employees (manager);`

`CREATE INDEX idx_facturas ON facturas (idCliente, fecha);`

`CREATE INDEX idx_alb ON albaran (fecha DESC, nombre);`

Cuando NO crear índices?

- ⇒ tablas pequeñas
- ⇒ tablas grandes y f ↓ inserción actualización
- ⇒ columnas a indexar → gran cantidad NULLS
- ⇒ columnas indexadas → manipuladas con f

Borrado de Índices



`DROP INDEX idx_clientes;`

`DROP INDEX IF EXISTS indiceFacturas;`

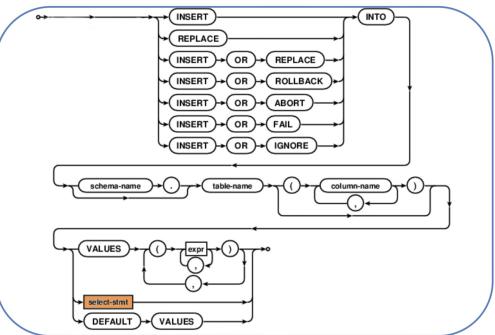
`DROP INDEX IF EXISTS miBaseDatos.miIndiceOrdenadores;`

4

Tema 3: Implementación BBDD

Manipulación de datos mediante SQL

Inserción



3 formas básicas de Inserción :

1) `VALUES` : inserta 1 o + registros en una tabla. Ej:

↳ Si se omite lista columnas / campos :

 nº valores insertados = nº de campos de la tabla
(debe ser)

↳ Si se especifica lista columnas / campos:

 valores deben coincidir en nº con lista

 (campos q. no aparecen en la lista → null)

2) `SELECT statements` : inserta 1 o varios registros cuyos valores → resultado consulta `SELECT` (coincidir G.nº)

3) `DEFAULT VALUES` : inserta nuevo registro.

 a cada campo → valor por defecto [nº null]

Ejemplos de Inserción

```
CREATE TABLE producto(  
    id integer PRIMARY KEY AUTOINCREMENT,  
    nombre text(20),  
    precio integer,  
    detalle text(10) DEFAULT 'OK');
```

```
INSERT INTO producto(id, nombre, precio, detalle)  
VALUES (1, 'Pancakes', 75, 'OK');
```

```
sqlite> .mode column  
sqlite> .header on
```

```
INSERT INTO producto(nombre, precio, detalle)  
VALUES ('Gulha', 55, 'Problems');
```

```
INSERT INTO producto VALUES(3, 'Pakora', 48, 'OK');
```

```
INSERT INTO producto(id, nombre) VALUES(4, 'Pizza');
```

```
sqlite> .nullvalue NULL
```

```
INSERT OR REPLACE INTO producto(id, nombre, precio, detalle)  
VALUES (4, 'Pizza', 200, 'OK');
```

Ejemplos de Inserción

```
CREATE TABLE producto(  
    id integer PRIMARY KEY AUTOINCREMENT,  
    nombre text(20),  
    precio integer,  
    detalle text(10) DEFAULT 'OK');
```

```
INSERT INTO producto(id, nombre, precio, detalle)  
VALUES (1, 'Pancakes', 75, 'OK');
```

```
sqlite> .mode column  
sqlite> .header on
```

```
INSERT INTO producto(nombre, precio, detalle)  
VALUES ('Gulha', 55, 'Problems');
```

```
INSERT INTO producto VALUES(3, 'Pakora', 48, 'OK');
```

```
INSERT INTO producto(id, nombre) VALUES(4, 'Pizza');
```

```
sqlite> .nullvalue NULL
```

```
INSERT OR REPLACE INTO producto(id, nombre, precio, detalle)  
VALUES (4, 'Pizza', 200, 'OK');
```

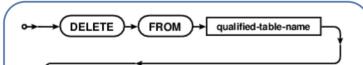


DROP VIEW: borra vista (no datos)

```
DROP VIEW empleados;
```

```
DROP VIEW IF EXISTS empresal.empleados;
```

Borrado



```
DELETE FROM employees;
```

```
DELETE FROM employees WHERE name = 'Peter';
```

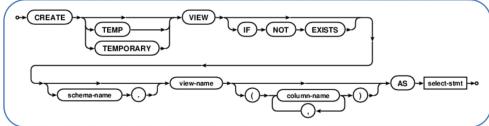
```
DELETE FROM employees WHERE salary > (SELECT AVG(salary) FROM employees WHERE job = 'Salesman');
```

Vistas

• Muestran resultados de la sentencia `SELECT`

• se almacenan como Query almacenada (Stored query)

Sintaxis Vistas:



```
CREATE VIEW cantantes AS SELECT * FROM artists;
```

```
CREATE TEMP VIEW cantantes2 AS SELECT ArtistId AS ID, Name AS Nombre FROM artists;
```

```
CREATE VIEW empleados AS  
SELECT EmployeeID AS ID, FirstName || " " || LastName AS "Nombre Completo",  
Email "Correo electrónico"  
FROM employees;
```

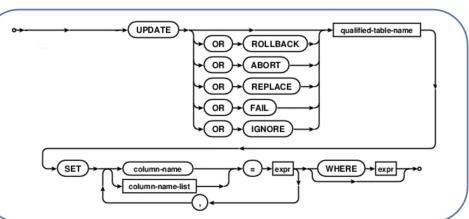
Borrado vistas

Tema 3: Implementación BRDD

Manipulación de datos mediante SQL

Actualización

cambiar valores existentes



Aumentar sueldo de todos los empleados: (+500 €)

```
UPDATE employees SET salary = salary + 500;
```

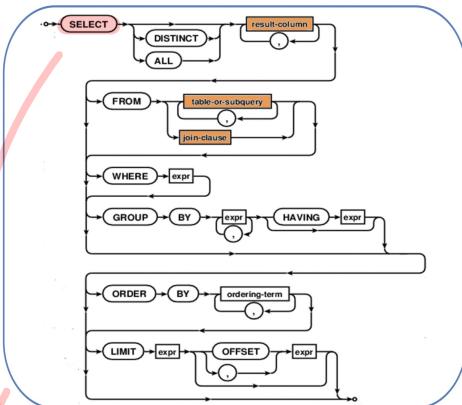
Cambiar ubicación departamento 69 para q coincida con la del 30:

```
UPDATE departments
SET location = (SELECT location FROM departments WHERE dep_id = 30)
WHERE dep_id = 69;
```

prod_id	prod_name	prod_rate	prod_qty
1	Pancakes	75	OK
2	Gulles	65	Problems
3	Waffles	45	OK
4	Pizza	200	OK
5	Fudge	100	OK
6	Candy	25	Not OK
7	Chocolate	150	OK

```
UPDATE prod_mast SET prod_gc='OK'
WHERE prod_gc>'OK';
```

Consulta



Cláusula	Acción a realizar	Valor de entrada
DISTINCT	Filtrar los resultados	Lista de columnas
FROM	Uniones (joins)	Lista de tablas
WHERE	Filtrar los resultados	Expresión o condición
GROUP BY	Agrupar los resultados	Lista de columnas
HAVING	Filtrar los resultados	Expresión o condición
ORDER BY	Ordenar los resultados	Lista de columnas
LIMIT	Filtrar los resultados	Valor entero
OFFSET	Filtrar los resultados	Valor entero

Ejemplo de Consultas:

Lista de todos los empleados con sus datos asociados

```
SELECT * FROM employees;
```

Lista empleados >> solo nombres y managers

```
SELECT name, manager FROM employees;
```

Lista nombre empleados >> ordenados por nombre Manager

```
SELECT name FROM employees ORDER BY manager;
```

Mostrar todos los managers

```
SELECT name, manager, dep_id FROM employees ORDER BY 2, 3 DESC;
```

Lista de nombre, manager y departamento de todos

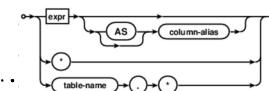
```
SELECT DISTINCT manager FROM employees;
```

EXPRESIONES

Realizar operaciones sobre los datos de consulta

Operandos: columnas, constantes, ...

Operadores: + ; - ; * ; /



```
SELECT nombre, salario * 12 "Salario anual" FROM empleados;
```

```
SELECT nombre, salario + bonus AS "Salario con bonus" FROM empleados;
```

6

TEMA 3: IMPLEMENTACIÓN BBDD

Manipulación de datos mediante SQL

(Consulta)

Predicados

• Expresión lógica sobre valores de columnas \rightarrow TRUE, FALSE, UNKNOWN

PDL → Cláusulas CHECK

PDM → Cláusulas CHECK y HAVING

• Predicados básicos: =, <, >, <=, >=

Muestra empleados contratados antes del 1-enero-1982

```
SELECT nombre FROM empleados WHERE fechaContrato < "1982-01-01";
```

PREDICADOS COMPLEJOS

combinaciones predicados con AND, OR, NOT

```
SELECT nombre FROM empleados WHERE puesto = "vendedor" AND salario > 1500;
```

comprobación de valor nulo:

- ↳ nombre_columna IS [NOT] NULL
- ↳ elimina reg con valor \leq NULL
- ↳ ej: empleados con bonificación

```
SELECT nombre FROM empleados WHERE bonus IS NOT NULL;
```

comprobación pertenencia a un conjunto:

- ↳ expresión [NOT] IN (valor-1 [, valor-2] ...)
- ↳ comproueba si \in lista de valores (después del IN)
- ↳ mostrar empleados llamados "Julio", "Ana" e "Andrés"

```
SELECT * FROM empleados WHERE nombre IN ("Julio", "Ana", "Andrés");
```

Cláusula GROUP BY → agrupar filas en una única fila

```
... GROUP BY nombre_columna1 [, nombre_columna2] ...
```

Mostrar los sueldos mínimo y máximo de los empleados, agrupados por puesto

```
SELECT puesto, MIN(salario), MAX(salario) FROM empleados GROUP BY puesto;
```

Calcular el número de empleados de cada departamento, que tienen un sueldo superior a 1500

```
SELECT dep_id, COUNT(nombre) FROM empleados WHERE salario > 1500 GROUP BY dep_id;
```

Cláusula HAVING → descartar grupos

```
HAVING expresión ...
```

Mostrar el puesto y número de empleados asignados al puesto, siempre que haya 5 o más personas con el mismo puesto

```
SELECT puesto, COUNT(*) FROM empleados GROUP BY puesto HAVING COUNT(*) >= 5;
```

Comparación Between-AND

- ↳ expresión [NOT] BETWEEN expresión2 AND expresión3
- ↳ comprueba si el valor está \in no comprendido entre otros dos
- ↳ Mostrar empleados : sueldo entre 2000 y 3000

```
SELECT nombre FROM empleados WHERE sueldo BETWEEN 2000 AND 3000;
```

Comparación LIKE

- ↳ nombre_columna [NOT] LIKE patrón
- ↳ reúntete a combinaciones caracteres q coincidan con el patrón especificado
 - ↳ comodines \rightarrow - "Cualquier carácter (solo uno)"
 - ↳ % "Cualquier orden de caracteres"
- ↳ EMPLEADOS \rightarrow 2º letra nombre = d

```
SELECT nombre FROM empleados WHERE nombre LIKE "_d%";
```

Cláusula ORDER BY → ordenar resultados de una consulta ASC DESC

Obtener todos los empleados ordenados por departamento y sueldo (mayor a menor)

```
SELECT * FROM empleados ORDER BY dep_id, sueldo DESC;
```

Cláusula LIMIT → limitar el num resultados

Obtener los nombres de 5 empleados, sin considerar los 4 primeros

```
SELECT nombre FROM empleados LIMIT 4, 5;
```

```
SELECT nombre FROM empleados LIMIT 5 OFFSET 4;
```

SUBCONSULTAS o SENTENCIAS SUBORDINADAS

- Mostrar los empleados cuyo salario sea superior a la media

```
SELECT nombre, salario FROM empleados WHERE salario > (SELECT AVG(salario) FROM empleados);
```

- Mostrar los empleados que trabajan en cualquier pueblo de la provincia de Madrid (28)

```
SELECT nombre FROM empleados WHERE pueblo_id IN (SELECT pueblo_id FROM pueblos WHERE provincias=28);
```

OPERADOR UNIÓN → combinar resultados de 2 o más SELECT

```
SELECT_1 UNION [ALL] SELECT_2 ...
```

• Por defecto → eliminan filas duplicadas (evitarlo → UNION ALL)

• Para usar UNION → todos SELECTS

= num columnas
= num expresiones columna
= tipo de datos
= orden

Obtener un identificador del empleado + puesto trabajo actuales y pasados:

```
SELECT empleado_id, puesto FROM empleados  
UNION  
SELECT empleado_id, puesto FROM empleados_histórico;
```

```
SELECT empleado_id, puesto FROM empleados  
UNION ALL  
SELECT empleado_id, puesto FROM empleados_histórico
```

7

TEMA 3: Implementación BBDD

Manipulación de datos mediante SQL

(Consulta)

Consulta sobre Varios Tablas

- 1) FROM
nombres de las tablas → separados por comas
[Una tabla puede aparecer más de una vez en FROM + cláusulas]

2) Consultas Correlacionadas

- tienen consultas subordinadas
- todas consultas → extraen datos de la misma tabla

¿Cuántos empleados trabajan en "Córdoba"?

```
SELECT COUNT(*) FROM empleados, departamentos  
WHERE empleados.dep_id = departamentos.id AND localidad = "Córdoba";
```

¿Qué empleados trabajan en "Salamanca"?

```
SELECT empleados.nombre FROM empleados, departamentos  
WHERE empleados.dep_id = departamentos.id AND localidad = "Salamanca";
```

Mostrar los nombres, puestos, departamentos y lugares de trabajo de todos los empleados

```
SELECT empleados.nombre, puesto, departamento.nombre, localidad  
FROM empleados, departamentos WHERE empleados.dep_id = departamentos.id;
```

Mostrar para cada empleado su nombre, el de su jefe, y cuánto gana su jefe

```
SELECT emps.nombre, emps.jefe, jefes.salario  
FROM empleados AS emps, empleados AS jefes WHERE jefes.nombre = emps.jefe;
```

Empleado (nombre, apellidos, ...)

↳ SELECT nombre || " " || apellidos AS Usuario

↓

usuario

Paloma Sánchez

José ...

↳ group.concat(id) ... Group by name

Paloma 123
1,4,149,372,3,56

Operaciones de Reunión (JOIN)

- Combinar registros de dos o más tablas → usando valores que tienen en común ambas
- JOIN más eficiente → productos cartesianos (UNION, ...)
- 4 TIPOS de JOIN

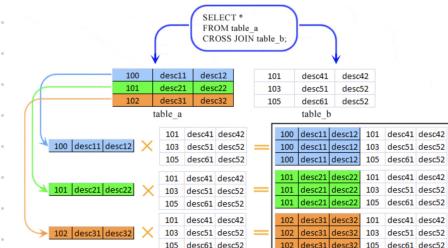


[En SQLite 3: CROSS, INNER, LEFT OUTER → se define Natural JOIN]

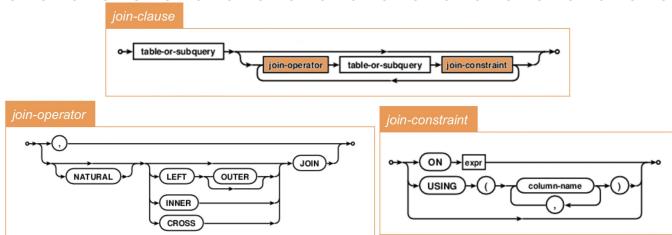
CROSS JOIN

Se empieza: cada fila 1º tabla → fila 2º tabla.

Producto cartesiano



SELECT * FROM empleados CROSS JOIN departamentos;



INNER JOIN

combinación de las columnas de las dos tablas → cumple cond de ON



• ¿Cuántos empleados trabajan en "Córdoba"?

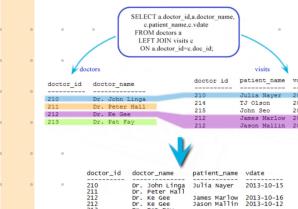
```
SELECT COUNT(*) FROM empleados INNER JOIN departamentos  
ON empleados.dep_id = departamentos.id WHERE localidad = "Córdoba";
```

• Mostrar los nombres, puestos, departamentos y lugares de trabajo de todos los empleados

```
SELECT empleados.nombre, puesto, departamento.nombre, localidad  
FROM empleados LEFT JOIN departamentos ON empleados.dep_id = departamentos.id;
```

LEFT OUTER JOIN

resultado: todos los valores tabla 1
+ empalmamientos → null
+ valores 2º tabla



otra tabla