SQL

Adolfo Sanz De Diego

Noviembre 2016

Table of Contents

# Acerca de

## Autor

* **Adolfo Sanz De Diego**
  + Blog: [asanzdiego.blogspot.com.es](http://asanzdiego.blogspot.com.es/)
  + Correo: [asanzdiego@gmail.com](mailto:asanzdiego@gmail.com)
  + GitHub: [github.com/asanzdiego](http://github.com/asanzdiego)
  + Twitter: [twitter.com/asanzdiego](http://twitter.com/asanzdiego)
  + LinkedIn: [in/asanzdiego](http://www.linkedin.com/in/asanzdiego)
  + SlideShare: [slideshare.net/asanzdiego](http://www.slideshare.net/asanzdiego/)

## Licencia

* **Copyright:**
  + Antonio Sarasa Cabezuelo <[antoniosarasa@campusciff.net](mailto:antoniosarasa@campusciff.net)>

## Fuente

* Las slides y sus fuentes las podéis encontrar en:
  + <https://github.com/asanzdiego/curso-intro-linux-web-sql-2016>

# SQLite

## ¿Qué es?

* **SQLite** es un sistema de gestión de bases de datos relacional, contenida en una relativamente pequeña (~275 kiB) biblioteca escrita en C.
* SQLite es un proyecto de dominio público creado por D. Richard Hipp.

## Instalación

* En Ubuntu:

sudo apt install sqlite3

# SQLiteBrowser

## ¿Qué es?

* Es una **interfaz gráfica** para el manejo de SQLite.
* Tiene una Licencia GPL (Software Libre)

## Instalación

* En Ubuntu:

sudo apt install sqlitebrowser

# Creación de tablas

## CREATE TABLE

* Para crear una tabla se utiliza la sentencia **CREATE TABLE**:

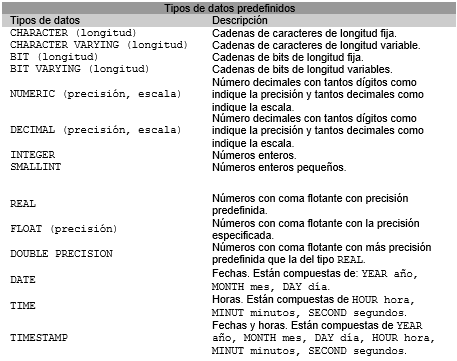
CREATE TABLE nombre\_tabla  
( definicion\_columna[, definicion\_columna...]  
[, restricciones\_tabla]);

## Consideraciones

* La definición de una columna consta del nombre de la columna, un tipo de datos predefinido, un conjunto de definiciones por defecto y restricciones de columna.

## Tipos de datos

* Los **principales tipos de datos** predefinidos en SQL que pueden asociarse a una columna son:



## Valores por defecto

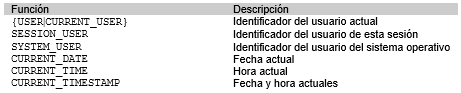
* Se pueden especificar valores por defecto mediante la sentencia:

DEFAULT (literal|función|NULL)

- Si se elige la \*\*opción NULL\*\*, entonces indica que la

columna debe admitir valores nulos. - Si se elige la **opción literal**, entonces indica que la columna tomará el valor indicado por el literal. - Si se elige la **opción función**, se indicará alguna de las funciones siguientes.

## Funciones

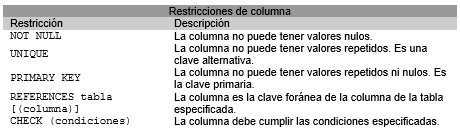


## Restricciones de columna

* Se pueden definir **restricciones sobre las columnas** de la siguiente forma:

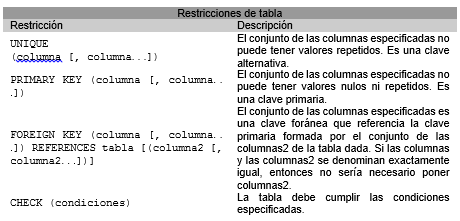
CONSTRAINT nombre\_restricción [CHECK(condiciones)]

## Lista restricciones columna



## Restricciones tabla

* Se pueden especificar **restricciones sobre toda la tabla**:



## Ejemplo 1

CREATE TABLE sucursal  
(nombre\_sucursal VARCHAR2(15) CONSTRAINT suc\_PK PRIMARY KEY,  
ciudad CHAR(20) NOT NULL CONSTRAINT cl\_UK UNIQUE,  
activos NUMBER(12,2) default 0);

## Ejemplo 2

CREATE TABLE cliente  
(dni VARCHAR2(9) NOT NULL,  
nombre\_cliente CHAR(35) NOT NULL,  
domicilio CHAR(50) NOT NULL,  
CONSTRAINT cl\_PK PRIMARY KEY (dni));

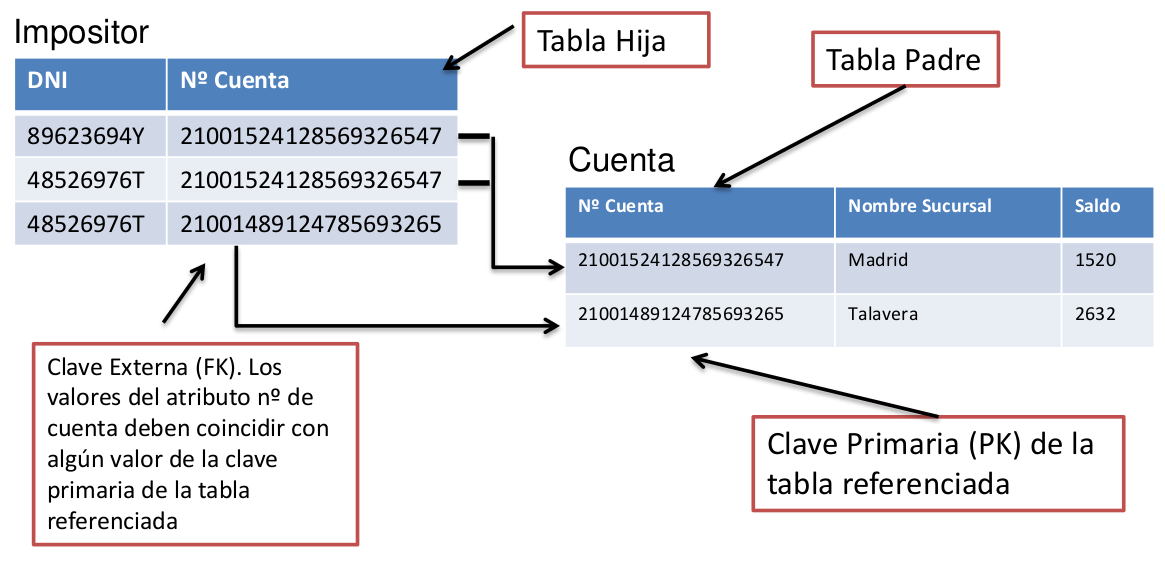
## Ejemplo 3

CREATE TABLE cuenta  
(numero\_cuenta CHAR (20) PRIMARY KEY,  
nombre\_sucursal char(15)  
REFERENCES sucursal,  
saldo NUMBER(12,2) default 100,  
CONSTRAINT imp\_minimo CHECK(saldo >=100))

## Ejemplo 4

CREATE TABLE impositor  
(dni CHAR(9) CONSTRAINT imp\_dni\_FK  
REFERENCES cliente,  
numero\_cuenta CHAR(20) NOT NULL,  
CONSTRAINT imp\_PK PRIMARY KEY (dni, numero\_cuenta),  
CONSTRAINT imp\_ct\_FK FOREIGN KEY (numero\_cuenta) REFERENCES cuenta)

## PK y FK



## Politicas de borrado

* Cuando se define una clave foránea se puede especificar las políticas de borrado y modificación de filas que tienen una clave primaria referenciada por claves foráneas de la siguiente forma:

FOREIGN KEY clave\_secundaria REFERENCES nombre\_tabla [(clave\_primaria)]  
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]  
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

* **NO ACTION** impide realizar alguna acción sobre un valor de clave primaria si en la tabla referenciada hay una valor de clave foránea relacionado.
* **CASCADE** representa la actualización en cascada. Borra o actualiza el registro en la tabla referenciada y en la tabla actual.
* **SET NULL** borra o actualiza el registro en la tabla referenciada y establece en NULL la/s columna/s de clave foránea en la tabla actual.
* **SET DEFAULT** indica que se ponga el valor especificado por defecto.

## Ejemplo 3 actualizado

CREATE TABLE cuenta  
(numero\_cuenta CHAR (20) PRIMARY KEY,  
nombre\_sucursal char(15)  
CONSTRAINT ct\_FK REFERENCES sucursal on delete set null,  
saldo NUMBER(12,2) default 100,  
CONSTRAINT imp\_minimo CHECK(saldo >=100))

## Ejemplo 4 actualizado

CREATE TABLE impositor  
(dni CHAR(9) CONSTRAINT imp\_dni\_FK  
REFERENCES cliente on delete cascade,  
numero\_cuenta CHAR(20),  
CONSTRAINT imp\_PK PRIMARY KEY (dni, numero\_cuenta),  
CONSTRAINT imp\_ct\_FK FOREIGN KEY (numero\_cuenta)  
REFERENCES cuenta on delete cascade)

## ALTER TABLE

* Para modificar una tabla se utiliza **ALTER TABLE**:

ALTER TABLE nombre\_tabla  
{accion\_modificar\_columna|accion\_modificar\_restriccion\_tabla};

## Añadir columna

* **Añadir columna** a una tabla.

ALTER TABLE nombre\_tabla  
ADD nombre\_columna TIPO [propiedades]

## Eliminar columna

* **Eliminar columna** de una tabla.

ALTER TABLE nombre\_tabla  
DROP COLUMN nombre\_columna

## Modificar columna

* **Modificar columna** de una tabla.

ALTER TABLE nombre\_tabla  
MODIFY (nombre\_columna TIPO [propiedades])

## Renombrar columna

* **Renombrar columna** de una tabla.

ALTER TABLE nombre\_tabla  
RENAME COLUMN nombre\_columna\_1 TO nombre\_columna\_2

## Añadir restricción

* **Añadir restricciones** a una tabla.

ALTER TABLE nombre\_tabla  
ADD CONSTRAINT nombre\_restriccion TIPO (columnas)

## Eliminar restricción

* **Eliminar restricciones** de una tabla.

ALTER TABLE nombre\_tabla  
DROP {PRIMARY KEY|UNIQUE(columnas)|CONSTRAINT nombre\_restriccion [CASCADE]}

* La opción CASCADE hace que se eliminen las restricciones de integridad que dependen de la eliminada.

## Ejemplo modificación

ALTER TABLE cuenta ADD comision NUMBER(4,2);  
ALTER TABLE cuenta ADD fecha\_apertura DATE;  
ALTER TABLE cuenta DROP COLUMN nombre\_sucursal;  
ALTER TABLE cuenta MODIFY comision DEFAULT 1.5;  
ALTER TABLE cliente MODIFY nombre\_cliente NULL;  
ALTER TABLE sucursal ADD CONSTRAINT cd\_UK UNIQUE(ciudad);

## Borrado de tablas

* Para **borrar una tabla** se utiliza la sentencia:

DROP TABLE nombre\_tabla {RESTRICT|CASCADE}

* **RESTRICT** indica que la tabla no se borrará si está referenciada.
* **CASCADE** indica que todo lo que referencie a la tabla se borrará con ésta.

## Renombrar tabla

* Para **renombrar una tabla** se utiliza la sentencia:

RENAME nombre\_tabla\_1 TO nombre\_tabla\_2

## Borrar contenido

* Para **borrar el contenido de una tabla** se utiliza la sentencia:

TRUNCATE TABLE nombre\_tabla

## Índices

* Los índices permiten que las bases de datos **aceleren las operaciones de consulta y ordenación** sobre los campos a los que el índice hace referencia.

## Indices implícitos

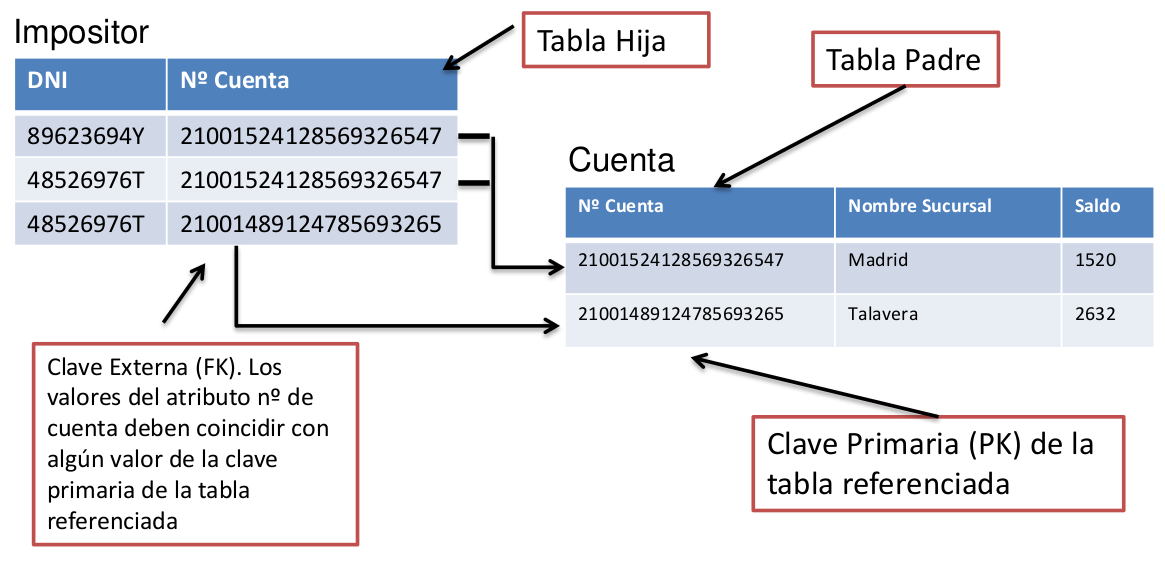
* La mayoría de los índices se crean de manera implícita, como consecuencia de las **restricciones PRIMARY KEY y UNIQUE**.

## Indices explícitos

* Se pueden crear explícitamente para aquellos campos sobre los cuales se realizarán **búsquedas** e instrucciones de **ordenación** frecuente.

CREATE [unique] INDEX nombre\_indice  
ON nombre\_tabla (col1,...,colk);

## Ejemplo índices



# Operaciones actualización

## INSERT INTO

* Para poder introducir datos en una tabla se usa **INSERT INTO**.

INSERT INTO nombre\_tabla [(nombres\_columnas)]  
{VALUES ({v1|DEFAULT|NULL}, ...,  
{vn/DEFAULT/NULL})|<consulta>};

## Consideraciones

* Los valores v1, v2, ..., vn se deben corresponder con las columnas de la tabla especificada y deben estar en el mismo orden.
* También es posible especificar el nombre de las columnas de la tabla. En este último caso, los valores se deben disponer de forma coherente con el nuevo orden.
* Si se quiere especificar que un valor por omisión se usa la palabra reservada DEFAULT, y si se trata de un valor nulo se usa la palabra reservada NULL.

## Insertar más de una fila

* Observar que para insertar más de una fila con una sola sentencia, se deben obtener los datos mediante una consulta a otras tablas.

## Ejemplo INSERT INTO 1

* Por ejemplo si se quiere insertar en una tabla clientes que tiene las columnas: nif, nombre\_cliente, codigo\_cliente, telefono, direccion, ciudad, se podría hacer de dos formas:

INSERT INTO clientes  
VALUES  
(10, 'Mercadona', '122233444-C', 'Gran vida 8', 'Madrid', DEFAULT);

## Ejemplo INSERT INTO 2

INSERT INTO clientes  
(nif, nombre\_cliente, codigo\_cliente, telefono, direccion, ciudad)  
VALUES  
('122233444-C', 'Mercadona', 10, DEFAULT, 'Gran vida 8', 'Madrid');

## DELETE

* Para **borrar valores de algunas filas** de una tabla se usa la sentencia DELETE:

DELETE FROM nombre\_tabla [WHERE condiciones];

## Consideraciones

* Observar que **si no se utiliza la clausula WHERE se borran todas las filas de la tabla**, en cambio si se utiliza WHERE entonces solo se borran aquellas filas que cumplen las condiciones especificadas.

## Ejemplo DELETE 1

* Por ejemplo si se quieren borrar todas las filas de la tabla proyectos se usaría la sentencia:

DELETE FROM proyectos;

## Ejemplo DELETE 2

* Si solo se quieren borrar las filas de la tabla en las que el valor de la columna cliente vale 12, entonces se usaría la sentencia:

DELETE FROM proyectos  
WHERE codigo\_cliente = 12;

## Ejemplo DELETE 3

* La clausula WHERE admite consultas anidadas como por ejemplo la consulta que quiere borrar todos los clientes que tengan un préstamo no registrado en la relación préstamo.

DELETE  
FROM clientes  
WHERE clientes.numero\_prestamos NOT IN (  
 SELECT numero\_prestamos  
 FROM prestamos);

## UPDATE

* Para modificar los valores de algunas filas de una tabla se usa la sentencia **UPDATE**:

UPDATE nombre\_tabla  
SET nombre\_columna = {expresión|DEFAULT|NULL}  
[, nombre\_columna = {expresión|DEFAULT|NULL} ...]  
WHERE condiciones;

## Consideraciones

* La cláusula SET indica qué columna modificar y los valores que puede recibir, y la cláusula WHERE especifica qué filas deben actualizarse.
* La parte WHERE es opcional y, si no se especifica, se actualizarán todas las tuplas de la tabla.

## Ejemplo UPDATE 1

* Por ejemplo si se quiere inicializar el sueldo de todos los empleados del proyecto 2 en 500 euros:

UPDATE empleados SET sueldo = 500  
WHERE numero\_proyecto = 2;

## Ejemplo UPDATE 2

* La clausula WHERE admite consultas anidadas como por ejemplo la siguiente consulta que quiere modificar todos los prestamos cuya sucursal hay sido cerrada a la sucursal 'Centro'.

UPDATE prestamos  
SET sucursal= 'Centro'  
WHERE sucursal IN (  
 SELECT sucursal  
 FROM sucursales\_cerradas);

# Consultas básicas

## SELECT

* Para hacer consultas sobre una tabla se utiliza **SELECT**:

SELECT nombre\_columna\_1 [[AS] columna\_renombrada\_1]  
[,nombre\_columna\_N [[AS] columna\_renombrada\_2]...]  
FROM nombre\_tabla [[AS] tabla\_renombrada];

## AS (renombrar)

* La palabra clave **AS permite renombrar** las columnas que se quieren seleccionar o las tablas que se quieren consultar. Esta palabra es opcional.

## Ejemplo sencillo

* Por ejemplo si queremos seleccionar las columnas código, nombre, dirección y ciudad de la tabla clientes usaríamos la sentencia:

SELECT codigo\_cliente, nombre\_cliente, direccion, ciudad  
FROM clientes;

## Ejemplo \*

* Se usa el símbolo \* si se quieren recuperar todas las columnas de la tabla:

SELECT \* FROM clientes;

## WHERE

* La clausula **WHERE** permite recuperar sólo aquellas filas que cumplen la condición especificada.

SELECT [DISTINCT|ALL] nombres\_columnas  
FROM nombre\_tabla [WHERE condiciones];

## DISTINCT

* La clausula **DISTINCT** permite indicar que nos muestre las filas resultantes sin repeticiones. La opción por defecto es **ALL** que indica que muestre todas las filas.

## Ejemplo DISTINCT

* Por ejemplo si se quieren recuperar los diferentes sueldos de la tabla empleados:

SELECT DISTINCT sueldo FROM empleados;

## Condiciones

* Para construir las condiciones de la clausula WHERE es necesario usar **operadores de comparación o lógicos**:
  + <(menor), >(mayor), =(igual),
  + <=(menor o igual), >=(mayor o igual), <>(distinto),
  + AND(conjunción de condiciones),
  + OR(disyunción de condiciones),
  + NOT(negación).

## Ejemplo condición

* Y si se quieren recuperar los empleados de la tabla empleados cuyo sueldo es mayor de 1000 euros:

SELECT \* FROM empleados WHERE sueldo> 1000;

## Subconsultas

* Una **subconsulta** es una consulta incluida dentro de otra consulta, y que aparece como parte de una cláusula WHERE.

## Ejemplo subconsulta

* Por ejemplo se quiere obtener los proyectos de la tabla proyectos que se corresponden con un cliente que tiene como NIF el número "444555-E":

SELECT \*  
FROM proyectos  
WHERE codigo\_cliente = (  
 SELECT codigo\_cliente  
 FROM clientes  
 WHERE nif="444555-E")

## Predicados

* En la condición que aparece en la clausula WHERE se pueden utilizar un conjunto de **predicados** predefinidos para construir las condiciones

## BETWEEN

* Expresa que se quiere encontrar un **valor entre unos límites** concretos:

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE nombre\_columna BETWEEN límite1 AND límite2;

## Ejemplo BETWEEN

* Por ejemplo se quieren recuperar todos los empleados cuyos sueldos están entre 1000 y 2000 euros:

SELECT codigo\_empleado  
FROM empleados  
WHERE sueldo BETWEEN 1000 and 2000;

## IN

* IN. Comprueba si un valor coincide con los

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE nombre\_columna [NOT] IN (valor1, ..., valorN);

## Ejemplo IN

* Por ejemplo se quieren recuperar todos los clientes que viven en Madrid y Zaragoza:

SELECT \*  
FROM clientes  
WHERE ciudad IN ('Madrid', 'Zaragoza');

## LIKE

* Comprueba si una columna de tipo carácter cumple una **condición determinada**.

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE nombre\_columna LIKE condición;

## Comodines LIKE

* Existen un conjunto de caracteres que actúan como **comodines**:
* El carácter \_ para cada representar un carácter individual.
* El carácter % para expresar una secuencia de caracteres incluido la secuencia vacía.

## Ejemplo LIKE 1

* Por ejemplo si se quieren recuperar los clientes cuya ciudad de residencia termina por la letra "d":

SELECT \* FROM clientes WHERE ciudad LIKE '%d';

## Ejemplo LIKE 2

* Y si se quiere refinar la consulta anterior y recuperar los clientes cuya ciudad de residencia termina por la letra "d" y el nombre de la ciudad tiene 5 letras:

SELECT \* FROM clientes WHERE ciudad LIKE '\_ \_ \_ \_ d';

## IS NULL

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE nombre\_columna IS [NOT] NULL;

## Ejemplo IS NULL

* Por ejemplo se quieren recuperar todos los clientes que no tienen un número de teléfono:

SELECT \*  
FROM clientes  
WHERE telefono IS NULL;

## EXISTS

* Comprueba si una consulta produce **algún resultado** (EXISTS) o no (NOT EXISTS):

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE [NOT] EXISTS subconsulta;

## Ejemplo EXISTS

* Por ejemplo se quieren recuperar todos los empleados que están asignados a algún proyecto:

SELECT \*  
FROM empleados  
WHERE EXISTS (  
 SELECT \*  
 FROM proyectos  
 WHERE codigo\_proyectoto = numero\_proyecto);

## ORDER BY

* Para ordenar los resultados de una consulta se utiliza la cláusula **ORDER BY**:

SELECT nombres\_columnas  
FROM nombre\_tabla  
[WHERE condiciones]  
ORDER BY nombre\_columna\_ordenar\_1 [DESC]  
[, nombre\_columna\_ordenar\_2 [DESC]...];

## DESC

* Por defecto los resultados se ordenan de manera **ascendente**. Así si queremos realizar una ordenación **descendente** se debe indicar usando la cláusula DESC.

## Ejemplo ORDER BY

* Por ejemplo si queremos ordenar los empleados por orden alfabético ascendente de a cuerdo a su nombre y descendente de acuerdo a su sueldo:

SELECT \*  
FROM empleados  
ORDER BY nombre\_empl, sueldo DESC;

# Consultas varias tablas

## Introducción

* En la cláusula FROM es posible **especificar más de una tabla** cuándo se quieren consultar columnas de tablas diferentes.

## Ejemplo JOIN 1

* Por ejemplo se quiere obtener el nif del cliente y el precio de los proyectos desarrollados para el cliente con código 30.

SELECT p.precio, c.nif  
FROM clientes c JOIN proyectos p  
ON c.codigo\_cliente = p.codigo\_cliente  
WHERE c.codigo\_cliente = 30;

## Ejemplo JOIN 1 alternativo

* Alternativamente se podría obtener con la siguiente consulta:

SELECT p.precio, c.nif  
FROM clientes c, proyectos p  
WHERE c.codigo\_cliente = p.codigo\_cliente  
AND c.codigo\_cliente = 30;

## Ejemplo JOIN 2

* Por ejemplo si se quieren los códigos de los proyectos que son más caros que el proyecto con código 30:

SELECT p1.codigo\_proyecto  
FROM proyectos p1 JOIN proyectos p2  
ON p1.precio > p2.precio  
WHERE p2.codigo\_proyecto = 30;

## Más de 2 tablas

* Para combinar **más de 2 tablas** basta añadirlas en el FROM de la consulta y establecer las relaciones necesarias en el WHERE, o bien combinar las tablas por pares de manera que la resultante es el primer componente del siguiente par.

## Ejemplo más de 2 tablas

* Por ejemplo si se quieren combinar las tablas empleados, proyectos y clientes:

SELECT \*  
FROM empleados e, proyectos p, clientes c  
WHERE e.numero\_proyecto = p.numero\_proyecto  
AND p.codigo\_cliente = c.codigo\_cliente;

## Intersección con IN

* La intersección se puede simular usando IN:

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE nombre\_columna IN (  
 SELECT nombre\_columna  
 FROM nombre\_tabla  
 [WHERE condiciones]);

## Intersección con EXISTS

* La intersección se puede simular usando EXISTS:

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE EXISTS (  
 SELECT \*  
 FROM nombre\_tabla  
 WHERE condiciones);

## Ejemplo INTERSECT

* Por ejemplo si se quiere saber las ciudades de los clientes en las que hay departamentos:

SELECT ciudad FROM clientes  
INTERSECT  
SELECT ciudad\_departamento FROM departamentos;

## Ejemplo intersección usando IN

SELECT c.ciudad  
FROM clientes c  
WHERE c.ciudad IN (  
 SELECT d.ciudad\_departamento  
 FROM departamentos d);

## Ejemplo intersección usando EXISTS

SELECT c.ciudad  
FROM clientes c  
WHERE EXISTS (  
 SELECT \*  
 FROM departamentos d  
 WHERE c.ciudad = d.ciudad\_departamento);

## Diferencia con NOT IN

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE nombre\_columna NOT IN (  
 SELECT nombre\_columna  
 FROM nombre\_tabla  
 [WHERE condiciones]);

## Diferencia con NOT EXISTS

SELECT nombres\_columnas  
FROM nombre\_tabla  
WHERE NOT EXISTS (  
 SELECT \*  
 FROM nombre\_tabla  
 [WHERE condiciones]);

## Ejemplo EXCEPT

* Por ejemplo si se quiere saber las ciudades de los clientes en las que no hay departamentos:

SELECT ciudad FROM clientes  
EXCEPT  
SELECT ciudad\_departamento FROM departamentos;

## Ejemplo diferencia con NOT IN

SELECT c.ciudad  
FROM clientes c  
WHERE c.ciudad NOT IN (  
 SELECT d.ciudad\_departamento  
 FROM departamentos d);

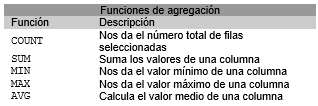
## Ejemplo diferencia con NOT EXISTS

SELECT c.ciudad  
FROM clientes c  
WHERE NOT EXISTS (  
 SELECT \*  
 FROM departamentos d  
 WHERE c.ciudad = d.ciudad\_departamento);

# Operaciones tablas

## Funciones de Agregación

* Las funciones de agregación son funciones que permiten realizar **operaciones sobre los datos de una columna**. Algunas funciones son las siguientes:



## COUNT(\*)

* En general, las funciones de agregación se aplican a una columna, excepto COUNT que se aplica a todas las columnas de las tablas seleccionadas. Se indica como **COUNT (\*)**.

## COUNT

* Sin embargo si se especifica **COUNT(distinct nombre columna)**, entonces sólo contará los valores que no son nulos ni repetidos, y se especifica **COUNT(columna)**, sólo contaría los valores que no son nulos.

## Ejemplo COUNT

* Por ejemplo si se quieren contar el número de clientes de la tabla clientes cuya ciudad es "Madrid":

SELECT COUNT(\*) AS numero\_clientes  
FROM clientes  
WHERE ciudad = 'Madrid';

## Agrupación de filas

* Al realizar una consulta, las filas se pueden agrupar de la siguiente manera:

SELECT nombres\_columnas  
FROM nombre\_tabla [WHERE condiciones]  
GROUP BY nombres\_columnas\_segun\_las\_cuales\_se\_quiere\_agrupar  
[HAVING condiciones\_por\_grupos]  
[ORDER BY nombre\_columna\_ordenacion [DESC] [, nombre\_columna\_ordenacion [DESC]...]];

## GROUP BY

* La cláusula **GROUP BY** permite agrupar las filas según las columnas indicadas, excepto aquellas afectadas por funciones de agregación.

## HAVING

* La cláusula **HAVING** especifica las condiciones para recuperar grupos de filas.

## Ejemplo agrupación de filas 1

* Por ejemplo si se quiere conocer el importe total de los proyectos agrupados por clientes:

SELECT codigo\_cliente, SUM(precio) AS importe  
FROM clientes  
GROUP BY codigo\_cliente;

## Ejemplo agrupación de filas 2

* Y si solo queremos aquellos clientes con un importe facturado mayor de 10000 euros:

SELECT codigo\_cliente  
FROM clientes  
GROUP BY codigo\_cliente  
HAVING SUM(precio)>10000

## Vistas

* Una vista es una **tabla ficticia** que se construye a partir de una consulta a una tabla real:

CREATE VIEW nombre\_vista [(lista\_columnas)]  
AS consulta\_sql [WITH CHECK OPTION];

## Borrar vistas

* Para **borrar una vista** se utiliza la sentencia:

DROP VIEW nombre\_vista (RESTRICT|CASCADE);

* La opción **RESTRICT** indica que la vista no se borrará si está referenciada.
* La opción **CASCADE** indica que todo lo que referencie a la vista se borrará con ésta.

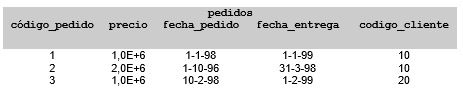
## Tabla clientes

* Tabla clientes:



## Tabla pedidos

* Tabla pedidos:



## Ejemplo Vistas

* Si se quiere crear una vista que indique para cada cliente el número de pedidos que tiene encargados el cliente, se definiría la vista:

CREATE VIEW pedidos\_por\_cliente (codigo\_cliente, num\_pedidos) AS  
 SELECT c.codigo\_cliente, COUNT(\\*)  
 FROM pedidos p, clientes c  
 WHERE p.codigo\_cliente = c.codigo\_cliente  
 GROUP BY c.codigo\_cliente;

## Vista pedidos por cliente

* Vista pedidos por cliente:

