SQL

ADOLFO SANZ DE DIEGO

OCTUBRE 2017

1 ACERCA DE

1.1 AUTOR

- **Adolfo Sanz De Diego**
  - Blog: asanzdiego.blogspot.com.es
  - Correo: asanzdiego@gmail.com
  - GitHub: github.com/asanzdiego
  - Twitter: twitter.com/asanzdiego
  - LinkedIn: in/asanzdiego
  - SlideShare: slideshare.net/asanzdiego

1.2 LICENCIA

- **Copyright:**
  - Antonio Sarasa Cabezuelo <antoniosarasa@campusciff.net>

1.3 FUENTE

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

2 MYSQL

2.1 ¿QUÉ ES?

- **MySQL** es un sistema de gestión de bases de datos relacional.
- Tiene una Licencia GPL (Software Libre)

2.2 INSTALACIÓN

- En Ubuntu: (acordaros de poner password de root y no olvidarla)

```
sudo apt install mysql-server
```

3 MYSQLWORKBENCH

3.1 ¿QUÉ ES?

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

3.2 INSTALACIÓN

- En Ubuntu:

```
sudo apt install mysql-workbench
```

4 CREACIÓN DE TABLAS

4.1 CREATE TABLE

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

```
CREATE TABLE nombre_tabla
( definicion_columna[, definicion_columna...]
[, restricciones_tabla]);
```

4.2 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.

4.3 TIPOS DE DATOS

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

Tipos de datos predefinidos	
Tipos de datos	Descripción
CHARACTER (longitud)	Cadenas de caracteres de longitud fija.
CHARACTER VARYING (longitud)	Cadenas de caracteres de longitud variable.
BIT (longitud)	Cadenas de bits de longitud fija.
BIT VARYING (longitud)	Cadenas de bits de longitud variables.
NUMERIC (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
DECIMAL (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
INTEGER	Números enteros.
SMALLINT	Números enteros pequeños.
REAL	Números con coma flotante con precisión predefinida.
FLOAT (precisión)	Números con coma flotante con la precisión especificada.
DOUBLE PRECISION	Números con coma flotante con más precisión predefinida que la del tipo REAL.
DATE	Fechas. Están compuestas de: YEAR año, MONTH mes, DAY día.
TIME	Horas. Están compuestas de HOUR hora, MINUT minutos, SECOND segundos.
TIMESTAMP	Fechas y horas. Están compuestas de YEAR año, MONTH mes, DAY día, HOUR hora, MINUT minutos, SECOND segundos.

Tipos de datos

4.4 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.

4.5 FUNCIONES

Función	Descripción
{USER|CURRENT_USER}	Identificador del usuario actual
SESSION_USER	Identificador del usuario de esta sesión
SYSTEM_USER	Identificador del usuario del sistema operativo
CURRENT_DATE	Fecha actual
CURRENT_TIME	Hora actual
CURRENT_TIMESTAMP	Fecha y hora actuales

Funciones

4.6 RESTRICCIONES DE COLUMNA

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

```
CONSTRAINT nombre_restricción [CHECK(condiciones)]
```

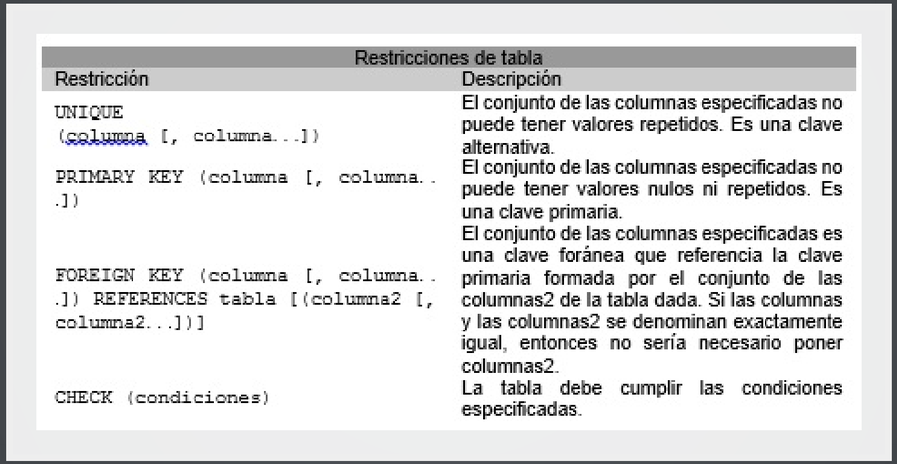
4.7 LISTA RESTRICCIONES COLUMNA

Restricciones de columna	
Restricción	Descripción
`NOT NULL`	La columna no puede tener valores nulos.
`UNIQUE`	La columna no puede tener valores repetidos. Es una clave alternativa.
`PRIMARY KEY`	La columna no puede tener valores repetidos ni nulos. Es la clave primaria.
`REFERENCES tabla [(columna)]`	La columna es la clave foránea de la columna de la tabla especificada.
`CHECK (condiciones)`	La columna debe cumplir las condiciones especificadas.

Restricciones columna

4.8 RESTRICCIONES TABLA

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



Restricciones de tabla	
Restricción	Descripción
`UNIQUE (columna [, columna...])`	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa.
`PRIMARY KEY (columna [, columna...])`	El conjunto de las columnas especificadas no puede tener valores nulos ni repetidos. Es una clave primaria.
`FOREIGN KEY (columna [, columna...]) REFERENCES tabla [(columna2 [, columna2...])]`	El conjunto de las columnas especificadas es una clave foránea que referencia la clave primaria formada por el conjunto de las columnas2 de la tabla dada. Si las columnas y las columnas2 se denominan exactamente igual, entonces no sería necesario poner columnas2.
`CHECK (condiciones)`	La tabla debe cumplir las condiciones especificadas.

Restricciones tabla

4.9 EJEMPLO 1

```
CREATE TABLE sucursal
(nombre_sucursal VARCHAR(15),
ciudad CHAR(20) NOT NULL,
activos DECIMAL(12,2) default 0,
CONSTRAINT suc_PK PRIMARY KEY (nombre_sucursal),
CONSTRAINT cl_UK UNIQUE (ciudad));
```

4.10 EJEMPLO 2

```
CREATE TABLE cliente
(dni VARCHAR(9) NOT NULL,
nombre_cliente CHAR(35) NOT NULL,
domicilio CHAR(50) NOT NULL,
CONSTRAINT cl_PK PRIMARY KEY (dni));
```

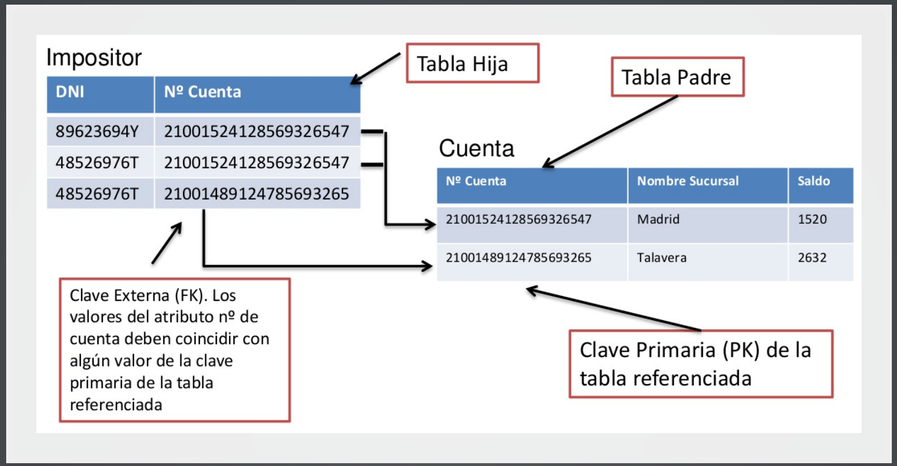
4.11 EJEMPLO 3

```
CREATE TABLE cuenta
(numero_cuenta CHAR (20) PRIMARY KEY,
nombre_sucursal char(15) REFERENCES sucursal,
saldo DECIMAL(12,2) default 100,
CONSTRAINT imp_minimo CHECK(saldo >=100))
```

4.12 EJEMPLO 4

```
CREATE TABLE impositor
(dni VARCHAR(9) REFERENCES cliente,
numero_cuenta CHAR(20) NOT NULL REFERENCES cuenta,
CONSTRAINT imp_PK PRIMARY KEY (dni, numero_cuenta))
```

4.13 PK Y FK



PK y FK

4.14 POLITICAS DE BORRADO (I)

- 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}]
```

4.15 POLITICAS DE BORRADO (II)

- **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.

4.16 EJEMPLO 3 ACTUALIZADO

```
CREATE TABLE cuenta
(numero_cuenta CHAR (20) PRIMARY KEY,
nombre_sucursal char(15) REFERENCES sucursal on delete set null,
saldo DECIMAL(12,2) default 100,
CONSTRAINT imp_minimo CHECK(saldo >=100))
```

4.17 EJEMPLO 4 ACTUALIZADO

```
CREATE TABLE impositor
(dni CHAR(9) REFERENCES cliente on delete cascade,
numero_cuenta CHAR(20) REFERENCES cuenta on delete cascade,
CONSTRAINT imp_PK PRIMARY KEY (dni, numero_cuenta))
```

4.18 ALTER TABLE

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

```
ALTER TABLE nombre_tabla
{accion_modificar_columna|accion_modificar_restriccion_tabla};
```

4.19 AÑADIR COLUMNA

- **Añadir columna** a una tabla.

```
ALTER TABLE nombre_tabla
ADD nombre_columna TIPO [propiedades]
```

4.20 ELIMINAR COLUMNA

- **Eliminar columna** de una tabla.

```
ALTER TABLE nombre_tabla
DROP COLUMN nombre_columna
```

4.21 MODIFICAR COLUMNA

- **Modificar columna** de una tabla.

```
ALTER TABLE nombre_tabla
MODIFY (nombre_columna TIPO [propiedades])
```

4.22 RENOMBRAR COLUMNA

- **Renombrar columna** de una tabla.

```
ALTER TABLE nombre_tabla
RENAME COLUMN nombre_columna_1 TO nombre_columna_2
```

4.23 AÑADIR RESTRICCIÓN

- **Añadir restricciones** a una tabla.

```
ALTER TABLE nombre_tabla
ADD CONSTRAINT nombre_restriccion TIPO (columnas)
```

4.24 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.

4.25 DESACTIVAR RESTRICCIONES

- **Desactivar restricciones** a una tabla.

```
ALTER TABLE nombre_tabla
DISABLE CONSTRAINT nombre_restriccion [CASCADE]
```

4.26 ACTIVAR RESTRICCIONES

- **Activar restricciones** a una tabla.

```
ALTER TABLE nombre_tabla
ENABLE CONSTRAINT nombre_restriccion
```

4.27 EJEMPLO MODIFICACIÓN

```
ALTER TABLE cuenta ADD comision DECIMAL(4,2);
ALTER TABLE cuenta ADD fecha_apertura DATE;
ALTER TABLE cuenta DROP COLUMN nombre_sucursal;
ALTER TABLE cuenta ALTER COLUMN comision SET DEFAULT 1.5;
ALTER TABLE cliente CHANGE COLUMN nombre_cliente nombre_cliente CHAR(35) NUL
ALTER TABLE sucursal ADD CONSTRAINT cd_UK UNIQUE(ciudad);
```

4.28 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.

4.29 DESCRIPCIÓN DE TABLA

- Para ver la **descripción de una tabla** se utiliza la sentencia:

```
DESCRIBE nombre_tabla
```

4.30 RENOMBRAR TABLA

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

```
RENAME nombre_tabla_1 TO nombre_tabla_2
```

4.31 BORRAR CONTENIDO

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

```
TRUNCATE TABLE nombre_tabla
```

4.32 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.

4.33 INDICES IMPLÍCITOS

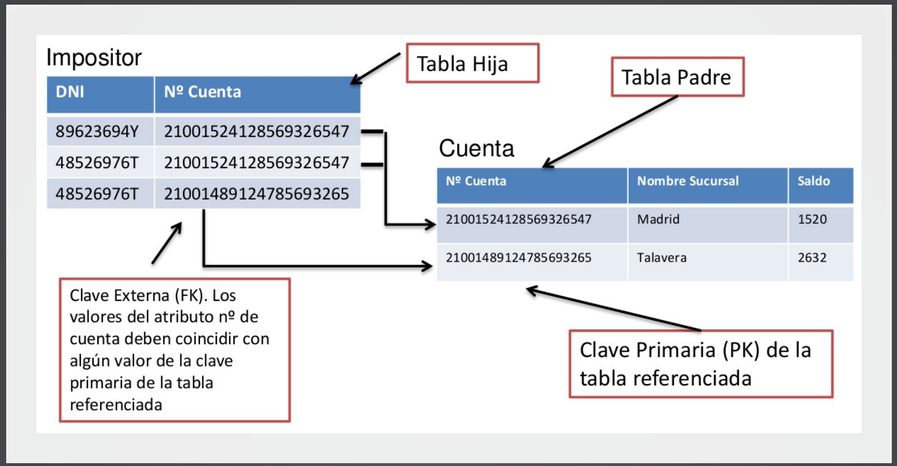
- La mayoría de los índices se crean de manera implícita, como consecuencia de las **restricciones PRIMARY KEY y UNIQUE**.

4.34 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);
```

4.35 EJEMPLO ÍNDICES



Ejemplo índices

5 OPERACIONES ACTUALIZACIÓN

5.1 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>};
```

5.2 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.

5.3 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.

5.4 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);
```

5.5 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');
```

5.6 DELETE

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

```
DELETE FROM nombre_tabla [WHERE condiciones];
```

5.7 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.

5.8 EJEMPLO DELETE 1

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

```
DELETE FROM proyectos;
```

5.9 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;
```

5.10 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);
```

5.11 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;
```

5.12 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.

5.13 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;
```

5.14 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);
```

6 CONSULTAS BÁSICAS

6.1 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];
```

6.2 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.

6.3 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;
```

6.4 EJEMPLO *

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

```
SELECT * FROM clientes;
```

6.5 WHERE

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

```
SELECT [DISTINCT|ALL] nombres_columnas
FROM nombre_tabla [WHERE condiciones];
```

6.6 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.

6.7 EJEMPLO DISTINCT

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

```
SELECT DISTINCT sueldo FROM empleados;
```

6.8 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).

6.9 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;
```

6.10 SUBCONSULTAS

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

6.11 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")
```

6.12 PREDICADOS

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

6.13 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;
```

6.14 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;
```

6.15 IN

- IN. Comprueba si un valor coincide con los **elementos de una lista** (IN) o no coincide(NOT IN):

```
SELECT nombres_columnas
FROM nombre_tabla
WHERE nombre_columna [NOT] IN (valor1, ..., valorN);
```

6.16 EJEMPLO IN

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

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

6.17 LIKE

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

```
SELECT nombres_columnas
FROM nombre_tabla
WHERE nombre_columna LIKE condición;
```

6.18 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.

6.19 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';
```

6.20 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';
```

6.21 IS NULL

- Comprueba si un **valor nulo** (IS NULL) o no lo es (IS NOT NULL):

```
SELECT nombres_columnas
FROM nombre_tabla
WHERE nombre_columna IS [NOT] NULL;
```

6.22 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;
```

6.23 EXISTS

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

```
SELECT nombres_columnas
FROM nombre_tabla
WHERE [NOT] EXISTS subconsulta;
```

6.24 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);
```

6.25 ANY/SOME/ALL

- Comprueba si todas (ALL) o algunas(SOME/ANY) de las filas de una columna cumplen las condiciones especificadas:

```
SELECT nombres_columnas
FROM nombre_tabla
WHERE nombre_columna operador_comparación {ALL|ANY|SOME} subconsulta;
```

6.26 EJEMPLO ALL

- Por ejemplo si se quiere recuperar todos los proyectos en los que los sueldos de todos los empleados asignados son menores que el precio del proyecto:

```
SELECT * FROM proyectos
WHERE precio > ALL (
  SELECT sueldo
  FROM empleados
  WHERE codigo_proyecto = numero_proyecto);
```

6.27 EJEMPLO SOME

- Si la condición se relaja, y sólo se pide que la condición sólo ocurra para algunos empleados, entonces sería:

```
SELECT * FROM proyectos
WHERE precio > SOME (
  SELECT sueldo
  FROM empleados
  WHERE codigo_proyecto = numero_proyecto);
```

6.28 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]...];
```

6.29 DESC

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

6.30 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;
```

7 CONSULTAS VARIAS TABLAS

7.1 INTRODUCCIÓN

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

7.2 COMBINACIÓN

- Se **simula una sola tabla** a partir de las tablas especificadas, haciendo coincidir los valores de las columnas relacionadas de las tablas.

```
SELECT nombres_columnas
FROM nombre_tabla_1 JOIN nombre_tabla_2
{ON condiciones | USING (nombre_columna [, nombre_columna...])}
[WHERE condiciones];
```

7.3 CONSIDERACIONES

- La opción ON permite expresar condiciones con cualquiera de los operadores de comparación sobre las columnas especificadas.
- Es posible utilizar una misma tabla dos veces usando alias diferentes para diferenciarlas.
- Puede ocurrir que las tablas consideradas tengan columnas con los mismos nombres. En este caso es obligatorio diferenciarlas especificando en cada columna a que tabla pertenecen.

7.4 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;
```

7.5 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;
```

7.6 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;
```

7.7 COMBINACIÓN NATURAL

- Consiste en una combinación en la que **se eliminan las columnas repetidas**.

```
SELECT nombres_columnas
FROM nombre_tabla_1 NATURAL JOIN nombre_tabla_2
[WHERE condiciones];
```

7.8 EJEMPLO NATURAL JOIN

- Por ejemplo si se quiere obtener los empleados cuyo departamento se encuentra situado en Madrid:

```sql
SELECT *
FROM empleados NATURAL JOIN departamentos
WHERE ciudad = 'Madrid';
```

7.9 EJEMPLO NATURAL JOIN ALTERNATIVO

- De forma equivalente se podría consultar:

```
SELECT *
FROM empleados JOIN departamentos
USING (nombre_dep, ciudad_departamento)
WHERE ciudad = 'Madrid';
```

7.10 COMBINACIÓN INTERNA

- Sólo se consideran las filas que tienen **valores idénticos en las columnas de las tablas** que compara.

```
SELECT nombres_columnas
FROM nombre_tabla_1 INNER JOIN nombre_tabla_2
{ON condiciones || USING (nombre_columna [,nombre_columna...])}
[WHERE condiciones];
```

7.11 COMBINACIÓN EXTERNA

- Se consideran los valores de la tabla derecha, de la izquierda o de ambas tablas.

```
SELECT nombres_columnas
FROM nombre_tabla_1 [LEFT|RIGHT|FULL] [OUTER] JOIN nombre_tabla_2
{ON condiciones| [USING (nombre_columna [,nombre_columna...])}
[WHERE condiciones];
```

7.12 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.

7.13 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;
```

7.14 EJEMPLO MÁS DE 2 TABLAS ALTERNATIVO

```
SELECT *
FROM (
  empleados JOIN proyectos
  ON numero_proyecto = codigo_proyecto)
    JOIN clientes ON codigo_cliente = codigo_cliente;
```

7.15 UNION

- Permite unir los resultados de 2 o más consultas.

```
SELECT nombres_columnas FROM nombre_tabla [WHERE condiciones]
UNION [ALL]
SELECT nombres_columnas FROM nombre_tabla [WHERE condiciones];
```

- Observar que la cláusula ALL indica si se quieren obtener todas las filas de la unión (incluidas las repetidas)

7.16 EJEMPLO UNION

- Por ejemplo si se quiere obtener todas las ciudades que aparecen en las tablas de la base de datos:

```
SELECT ciudad FROM clientes
UNION
SELECT ciudad_departamento FROM departamentos;
```

7.17 INTERSECT

- Permite hacer la **intersección** entre los resultados de 2 o más consultas.

```
SELECT nombres_columnas FROM nombre_tabla [WHERE condiciones]
INTERSECT [ALL]
SELECT nombres_columnas FROM nombre_tabla [WHERE condiciones];
```

- La cláusula ALL indica si se quieren obtener todas las filas de la intersección (incluidas las repetidas)

7.18 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]);
```

7.19 INTERSECCIÓN CON EXISTS

- La intersección se puede simular usando EXISTS:

```
SELECT nombres_columnas
FROM nombre_tabla
WHERE EXISTS (
  SELECT *
  FROM nombre_tabla
  WHERE condiciones);
```

7.20 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;
```

7.21 EJEMPLO INTERSECCIÓN USANDO IN

```
SELECT c.ciudad
FROM clientes c
WHERE c.ciudad IN (
  SELECT d.ciudad_departamento
  FROM departamentos d);
```

7.22 EJEMPLO INTERSECCIÓN USANDO EXISTS

```sql
SELECT c.ciudad
FROM clientes c
WHERE EXISTS (
  SELECT *
  FROM departamentos d
  WHERE c.ciudad = d.ciudad_departamento);
```

7.23 EXCEPT

- Permite hacer la **diferencia** entre los resultados de 2 o más consultas.

```
SELECT nombres_columnas FROM nombre_tabla [WHERE condiciones]
EXCEPT
SELECT nombres_columnas FROM nombre_tabla [WHERE condiciones];
```

7.24 DIFERENCIA CON NOT IN

```
SELECT nombres_columnas
FROM nombre_tabla
WHERE nombre_columna NOT IN (
  SELECT nombre_columna
  FROM nombre_tabla
  [WHERE condiciones]);
```

7.25 DIFERENCIA CON NOT EXISTS

```
SELECT nombres_columnas
FROM nombre_tabla
WHERE NOT EXISTS (
  SELECT *
  FROM nombre_tabla
  [WHERE condiciones]);
```

7.26 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;
```

7.27 EJEMPLO DIFERENCIA CON NOT IN

```sql
SELECT c.ciudad
FROM clientes c
WHERE c.ciudad NOT IN (
  SELECT d.ciudad_departamento
  FROM departamentos d);
```

7.28 EJEMPLO DIFERENCIA CON NOT EXISTS

```sql
SELECT c.ciudad
FROM clientes c
WHERE NOT EXISTS (
  SELECT *
  FROM departamentos d
  WHERE c.ciudad = d.ciudad_departamento);
```

8 OPERACIONES TABLAS

8.1 FUNCIONES DE AGREGACIÓN

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

Funciones de agregación	
Función	Descripción
COUNT	Nos da el número total de filas seleccionadas
SUM	Suma los valores de una columna
MIN	Nos da el valor mínimo de una columna
MAX	Nos da el valor máximo de una columna
AVG	Calcula el valor medio de una columna

Funciones de Agregación

8.2 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 (*)**.

8.3 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.

8.4 EJEMPLO COUNT

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

```sql
SELECT COUNT(*) AS numero_clientes
FROM clientes
WHERE ciudad = 'Madrid';
```

8.5 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 [DES
```

8.6 GROUP BY

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

8.7 HAVING

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

8.8 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;
```

8.9 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
```

8.10 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 [WITH CHECK OPTION];
```

8.11 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.

8.12 TABLA CLIENTES

- Tabla clientes:

clientes					
codigo_cli	nombre_cli	nif	dirección	ciudad	teléfono
10	Carrefour	38.567.893-C	Gran vía 11	Madrid	NULL
20	El Corte Inglés	38.123.898-E	Plaza de España 22	Zaragoza	976 45 56 78
30	Mercadona	36.432.127-A	Begoña, 33	Bilbao	940 34 56 90

Tabla clientes

8.13 TABLA PEDIDOS

- Tabla pedidos:

pedidos				
código_pedido	precio	fecha_pedido	fecha_entrega	codigo_cliente
1	1,0E+6	1-1-98	1-1-99	10
2	2,0E+6	1-10-96	31-3-98	10
3	1,0E+6	10-2-98	1-2-99	20

Tabla pedidos

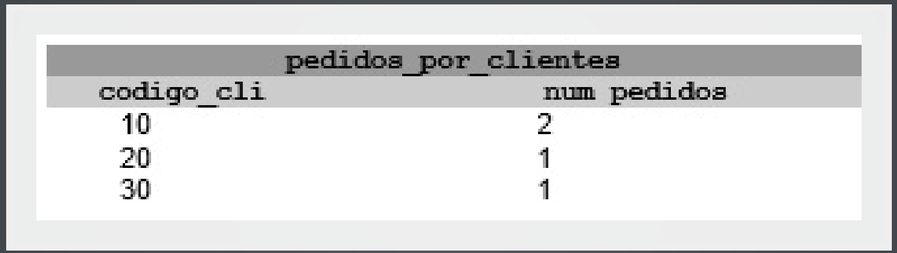
8.14 EJEMPLO VISTAS

- 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;
```

8.15 VISTA PEDIDOS POR CLIENTE

- Vista pedidos por cliente:



pedidos_por_clientes	
codigo_cli	num pedidos
10	2
20	1
30	1

Vista pedidos por cliente