Introducción a SQL

Antonio Sarasa Cabezuelo

Índice de contenidos

Introducción

SQL

- SQL es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales.
- Se caracteriza por:
  - Es un lenguaje declarativo basado en el álgebra relacional.
  - Está soportado por la mayoría de los sistemas relacionales comerciales.
  - Se puede utilizar de manera interactiva o a bien embebido en un programa.

SQL y Modelo relacional

- En el modelo relacional se estructura la información en base a los conceptos:
  - Relación.
  - Atributos.
  - Tuplas.
- En SQL se consideran concepto similares pero con una nomenclatura diferente:
  - Tablas.
  - Columnas.
  - Filas.

Creación/Eliminación de tablas

Creación de tablas

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

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

- 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:

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

- Para trabajar con el tiempo se usa la siguiente nomenclatura:
  - YEAR (0001..9999)
  - MONTH (01..12)
  - DAY (01..31)
  - HOUR (00..23)
  - MINUT (00..59)
  - SECOND (00..59.precisión)

Tipos de datos

- Por ejemplo:
  - Una columna fecha_nacimiento podría ser del tipo DATE y tomar el valor ‘1978-12-25’.
  - Una columna inicio_pelicula podría ser del tipo TIME y tomar el valor ‘17:15:00.000000’.
  - Una columna entrada_clase podría ser de tipo TIMESTAMP y tomar el valor ‘1998-7-8 9:30:05’.

Valores por defecto

- La opción **def_defecto** permite especificar valores por omisión mediante la sentencia:

DEFAULT (literal|función|**NULL**)

- donde:
  - 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.

Valores por defecto

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

Valores por defecto

- La opción **restricciones_dominio** tiene el siguiente formato:

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

Restricciones columnas

- Cuando se define una columna, además de especificar su nombre y tipo, se pueden establecer un conjunto de restricciones que siempre se tienen que cumplir:

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 tabla

- Cuando se han definido las columnas de una tabla, a continuación se pueden especificar restricciones sobre toda la tabla, que siempre se deberán cumplir:

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

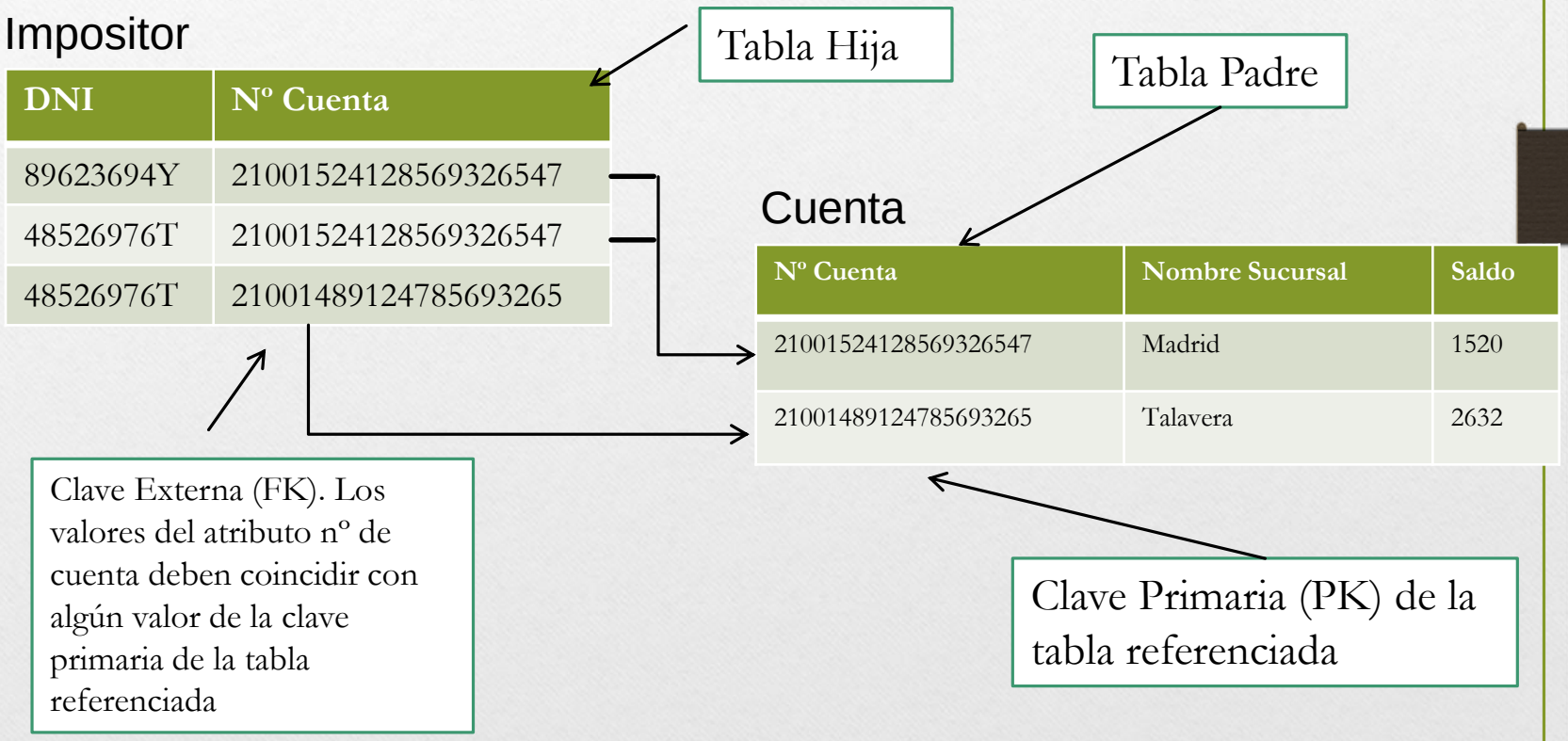
```
Create table sucursal
(nombre_sucursal VARCHAR2(15) CONSTRAINT suc_PK PRIMARY KEY,
 ciudad CHAR(20) NOT NULL CONSTRAINT cl_UK UNIQUE,
 activos NUMBER(12,2) default 0);

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));
```

Restricciones tabla

```
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))
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)
```

Definición de Datos - Tablas



Restricciones tabla

- 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** tabla [(clave_primaria)]

[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

Restricciones tabla

- Donde:
  - NO ACTION indica no realizar ninguna acción en el sentido de que un intento de borrar o actualizar un valor de clave primaria no será permitido 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 automáticamente borra o actualiza los registros coincidentes 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.

Restricciones tabla

```
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))

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)
```

Modificación de tablas

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

ALTER TABLE nombre_tabla
{acción_modificar_columna|acción_modif_restricción_tabla};

- **acción_modificar_columna** puede ser:

  {**ADD** [**COLUMN**] columna def_columna **|**

  ALTER [**COLUMN**] columna {**SET** def_defecto**|DROP DEFAULT**}**|**

  DROP [**COLUMN**] columna {**RESTRICT|CASCADE**}}

Modificación de tablas

- **acción_modif_restricción_tabla puede ser:**

  {ADD restricción|DROP CONSTRAINT restricción
  {RESTRICT|CASCADE}}

- Así pues las acciones de modificación que pueden realizarse sobre una tabla son:
  - Añadir una columna (ADD columna).
  - Modificar las definiciones por defecto de una columna (ALTER columna).
  - Borrar una columna (DROP columna).
  - Añadir alguna nueva restricción de tabla (ADD restricción).
  - Borrar alguna restricción de tabla (DROP CONSTRAINT restricción).

Definición de Datos

- *Añadir* atributos a una tabla.

  alter table *R* add Atributo Dominio [propiedades]

- *Eliminar* atributos de una tabla.

  alter table *R* drop COLUMN Atributo

  - *No se puede eliminar la única columna de una tabla.*
  - *Si la columna interviene en una constraint dará error:*
    - **alter table R drop Atributo CASCADE CONSTRAINTS**

- *Modificar* atributos de una tabla.

  alter table *R* modify (Atributo Dominio [propiedades])

- *Renombrar* atributos de una tabla.

  alter table *R* rename column Atributo1 to Atributo2

Definición de Datos

- *Añadir* restricciones a una tabla.

```
alter table R add CONSTRAINT nombre Tipo (columnas)
```

- *Eliminar* restricciones de una tabla.

```
alter table R drop {PRIMARY KEY|UNIQUE(campos)|
                    CONSTRAINT nombre [CASCADE]}
```

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

- *Desactivar* restricciones.

```
alter table R disable CONSTRAINT nombre [CASCADE]
```

- *Activar* restricciones.

```
alter table R enable CONSTRAINT nombre
```

Definición de Datos

```
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**:

  DROP TABLE nombre_tabla{**RESTRICT**|**CASCADE**};

- donde:
  - La opción **RESTRICT** indica que la tabla no se borrará si está referenciada,
  - La opción **CASCADE** indica que todo lo que referencie a la tabla se borrará con ésta.

Definición de Datos

- Descripción de una tabla.

  ```
  describe R
  ```

- Renombrar una tabla.

  ```
  rename R to S
  ```

- Borrar contenido de una tabla.

  ```
  truncate table R
  ```

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.
- La mayoría de los índices se crean de manera implícita, como consecuencia de las restricciones **PRIMARY KEY y UNIQUE.**
- Se pueden crear explícitamente para aquellos campos sobre los cuales se realizarán búsquedas e instrucciones de ordenación frecuente.

```
CREATE [unique] INDEX NombreIndice
ON NombreTabla(col1,...,colk);
```

Definición de Datos

Empleado

Cuenta

DNI	Nombre	Apellido	Dirección	Nº Empleado
3274623R	Antonio	López	Retama 14	RA45823633
27347234T	Marta	Sánchez	Loreto 134	PE74563669
85647456W	Antonio	López	Plaza Tornero 23	TY45236916
37562365F	María	Puente	Travesía del Reloj 1	MU45992355
34126455Y	Juan	Panero	Viviana 123	YR42326524

Varias filas pueden tomar los mismos valores en los atributos del índice

No hay dos filas con el mismo nº de registro

```
Create  index ID_Nombre  on  Empleado (Apellido, Nombre);
```

```
Create  unique index ID_Emp on Empleado (NºEmpleado);
```

Inserción de filas en una tabla

- Para poder consultar los datos de una base de datos hay que introducirlos con la sentencia **INSERT INTO VALUES:**

INSERT INTO nombre_tabla [(columnas)]

{**VALUES** ({v1|DEFAULT|NULL}, ...,
{vn/DEFAULT/NULL})|<consulta>};

- Los valores v1, v2, ..., vn se deben corresponder con las columnas de la tabla especificada y deben estar en el mismo orden, a menos que las volvamos a poner a continuación del nombre 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.

Inserción de filas en una tabla

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

Inserción de filas en una tabla

- Por ejemplo si se quiere insertar en una tabla clientes que tiene las columnas :nif, nombre_cli, codigo_cli, telefono, direccion, ciudad, se podría hacer de dos formas:

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

- O bien:

```
INSERT INTO clientes(nif, nombre_cli, codigo_cli, telefono, direccion, ciudad)
VALUES ('122233444-C', 'Mercadona', 10, DEFAULT, 'Gran vida 8', 'Madrid');
```

Inserción de filas en una tabla

- Insertar un préstamo en la relación Préstamo

```
INSERT INTO Prestamo
VALUES (‘Navacerrada’, ‘Pepe Pérez’, 125.000)
```

- También es posible obtener los datos mediante una consulta SELECT que actúe como proveedor de datos.

```
INSERT INTO Prestamo
SELECT * FROM Nuevos_Prestamos
```

Borrado de filas de una tabla

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

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

- Observar que si no se utiliza la claúsula **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.

Borrado de filas de una tabla.

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

```
DELETE FROM proyectos;
```

- Sin embargo 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;
```

Borrado de filas de una tabla.

- La clausula **WHERE** admite consultas anidadas
- Borrar todos los clientes que tengan un prestamo no registrado en la relación Préstamo.

```
DELETE
FROM Clientes
WHERE Clientes.NumPrestamo NOT IN
(SELECT NumPrestamo FROM Prestamo)
```

Modificación de filas de una tabla

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

```
UPDATE nombre_tabla
SET columna = {expresión|DEFAULT|NULL}
[, columna = {expr|DEFAULT|NULL} ...]
WHERE condiciones;
```

- La cláusula SET indica qué columna modificar y los valores que puede recibir, y la cláusula WHERE especifica qué filas deben actualizarse.

Modificación de filas de una tabla.

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

```
UPDATE empleados SET sueldo = 500
WHERE num_proyec = 2;
```

Modificación de Datos

- La parte **WHERE** es opcional y, si no se especifica, se actualizarán todas las tuplas de la tabla.

```
UPDATE Prestamo
SET importe=200.000
WHERE NumPrestamo=‘P-170’
```

Modificación de Datos

- La clausula **WHERE** admite consultas anidadas

Modificar todos los prestamos cuya sucursal hay sido cerrada a la sucursal 'Centro'.

```
UPDATE Prestamo
SET sucursal= 'Centro'
WHERE sucursal IN
        (SELECT sucursal
        FROM Sucursales_Cerradas)
```

Consulta de tablas

SELECT FROM

- Para hacer consultas sobre una tabla se utiliza la sentencia **SELECT:**

```
SELECT nombre_columna_a_seleccionar [[AS] col_renombrada]
[,nombre_columna_a_seleccionar [[AS] col_renombrada]...]
FROM tabla_a_consultar [[AS] tabla_renombrada];
```

- La palabra clave **AS** permite renombrar las columnas que se quieren seleccionar o las tablas que se quieren consultar. Esta palabra es opcional, y muchas veces se sustituye por un espacio en blanco.

SELECT FROM

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

```
SELECT codigo_cli, nombre_cli, direccion, ciudad FROM
clientes;
```

- Sin embargo si se quieren recuperar todas las columnas de la tabla se usa el símbolo "*", en vez de listar todas las columnas:

```
SELECT * FROM clientes;
```

SELECT FROM WHERE

- Si se quieren seleccionar que filas son recuperadas entonces hay que utilizar en la consulta SELECT la palabra reservada WHERE:

SELECT [DISTINCT|ALL] nombre_columnas_a_seleccionar
FROM tabla_a_consultar [**WHERE** condiciones];

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

SELECT FROM WHERE

- 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).

SELECT FROM WHERE

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

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

- 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 o HAVING(se verá a continuación).
- 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 código_cli
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 nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna BETWEEN límite1 AND límite2;
```

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

```
SELECT codigo_empl FROM empleados
WHERE sueldo BETWEEN 1000 and 2000;
```

Predicados

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

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna [NOT] IN (valor1, ..., valorN);
```

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

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

Predicados

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

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna LIKE condición;
```

- 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.

Predicados

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

```
SELECT * FROM clientes WHERE ciudad LIKE ‘%d’;
```

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 6 letras:

```
SELECT * FROM clientes WHERE ciudad LIKE ‘_ _ _ _ _d’;
```

Predicados

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

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna IS [NOT] NULL;
```

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

```
SELECT * FROM clientes WHERE teléfono IS NULL;
```

Predicados

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

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE [NOT] EXISTS subconsulta;
```

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

```
SELECT * FROM empleados
WHERE EXISTS (SELECT * FROM proyectos
        WHERE codigo_proyec = num_proyec);
```

Predicados

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

SELECT nombre_columnas_a seleccionar

FROM tabla_a_consultar

WHERE columna operador_comparación {**ALL**|**ANY**|**SOME**}subconsulta;

Predicados

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_proyec = num_proyec);
```

- 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_proyec = num_proyec);
```

Order by

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

```
SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
ORDER BY columna_según_la_cual_se_quiere_ordenar [DESC]
[, col_ordenación [DESC]...];
```

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

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;
```

Funciones de agregación

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

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

Funciones de agregación

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

```
SELECT COUNT(*) AS numero_clie FROM clientesWHERE ciudad = ‘Madrid’;
```

Agrupación de filas

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

SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar [**WHERE** condiciones]
GROUP BY columnas_según_las_cuales_se_quiere_agrupar
[**HAVING** condiciones_por_grupos]
[**ORDER BY** columna_ordenación [**DESC**] [, columna [**DESC**]...]];

- La cláusula **GROUP BY** permite agrupar las filas según las columnas indicadas, excepto aquellas afectadas por funciones de agregación.
- La cláusula **HAVING** especifica las condiciones para recuperar grupos de filas.

Agrupación de filas

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

```
SELECT código_cliente, SUM(precio) AS importe FROM clientes
GROUP BY codigo_cliente;
```

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

```
SELECT código_cliente FROM clientes
GROUP BY codigo_cliente
HAVING SUM(precio)>10000
```

Consultas sobre más de una tabla

Consulta de más de una tabla.

- En la cláusula FROM es posible especificar más de una tabla cuándo se quieren consultar columnas de tablas diferentes.
- Existen varios casos:
  - **Combinación**. Se crea una sola tabla a partir de las tablas especificadas, haciendo coincidir los valores de las columnas relacionadas de las tablas.

```
SELECT nombre_columnas_a_seleccionar
FROM tabla1 JOIN tabla2
{ON condiciones | USING (columna [, columna...])}
[WHERE condiciones];
```

Consulta de más de una tabla.

- 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.

Consulta de más de una tabla.

- 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_cli = p.codigo_cliente WHERE c.codigo_cli = 30;
```

- Alternativamente se podría obtener con la siguiente consulta:

```
SELECT p.precio, c.nif FROM clientes c, proyectos p
WHERE c.codigo_cli = p.codigo_cliente AND c.codigo_cli = 20;
```

Consulta de más de una tabla.

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

```
SELECT p1.codigo_proyec
FROM proyectos p1 JOIN proyectos p2 ON
p1.precio > p2.precio
WHERE p2.codigo_proyec = 30;
```

Consulta de más de una tabla.

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

```
SELECT nombre_columnas_a_seleccionar
FROM tabla1 NATURAL JOIN tabla2 [WHERE condiciones];
```

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

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

- De forma equivalente se podría consultar:

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

Consulta de más de una tabla.

- **Combinación interna vs externa.**
  - Interna(inner join). Sólo se consideran las filas que tienen valores idénticos en las columnas de las tablas que compara.

```
SELECT nombre_columnas_a_seleccionar
FROM t1 [NATURAL] [INNER] JOIN t2
{ON condiciones| |USING (columna [,columna...])}
[WHERE condiciones];
```

  - Externa(outer join). Se consideran los valores de la tabla derecha, de la izquierda o de ambas tablas.

```
SELECT nombre_columnas_a_seleccionar
FROM t1 [NATURAL] [LEFT|RIGHT|FULL] [OUTER] JOIN t2
{ON condiciones| [USING (columna [,columna...])}
[WHERE condiciones];
```

Consulta de más de una tabla.

- **Combinación de 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.
- Por ejemplo si se quieren combinar las tablas empleados, proyectos y clientes:

```
SELECT * FROM empleados, proyectos, clientes
WHERE num_proyec = codigo_proyec AND codigo_cliente = codigo_cli;
```

O bien:

```
SELECT * FROM (empleados JOIN proyectos ON num_proyec = codigo_proyec)JOIN clientes ON codigo_cliente = codigo_cli;
```

Operaciones sobre consultas

Operaciones entre consultas

- Entre 2 tablas se pueden definir las siguientes operaciones:
  - **Unión.** Permite unir los resultados de 2 o más consultas.

    SELECT columnas **FROM** tabla [**WHERE** condiciones]

    UNION [**ALL**]

    SELECT columnas **FROM** tabla[**WHERE** condiciones];

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

Operaciones entre consultas

- 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_dep FROM departamentos;
```

Operaciones entre consultas

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

```
SELECT columnas FROM tabla [WHERE condiciones]
INTERSECT [ALL]
SELECT columnas FROM tabla [WHERE condiciones];
```

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

Operaciones entre consultas

- Se puede simular usando:
  - **IN**

```
SELECT columnas FROM tabla WHERE columna
IN (SELECT columna FROM tabla [WHERE condiciones]);
```

  - **EXISTS**

```
SELECT columnas FROM tabla WHERE EXISTS
(SELECT *FROM tabla WHERE condiciones);
```

Operaciones entre consultas

- Por ejemplo si se quiere saber las ciudades de los clientes en las que hay departamentos:
  - Usando la intersección.

```
SELECT ciudad FROM clientes INTERSECT
SELECT ciudad_dep FROM departamentos;
```

  - Usando IN

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

  - Usando EXISTS

```
SELECT c.ciudad FROM clientes c WHERE EXISTS (SELECT *
FROM departamentos d WHERE c.ciudad = d.ciudad_dep;
```

Operaciones entre consultas

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

  SELECT columnas **FROM** tabla [**WHERE** condiciones]
  EXCEPT [ALL]
  SELECT columnas **FROM** tabla [**WHERE** condiciones];

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

Operaciones entre consultas

- Se puede simular usando:
  - **NOT IN**

    SELECT columnas **FROM** tabla **WHERE** columna **NOT IN** (**SELECT** columna **FROM** tabla [**WHERE** condiciones]);

  - **NOT EXISTS**

    SELECT columnas **FROM** tabla **WHERE NOT EXISTS** (**SELECT** ***FROM** tabla **WHERE** condiciones);

Operaciones entre consultas

- Por ejemplo si se quiere saber las ciudades de los clientes en las que no hay departamentos:
  - Usando la intersección.

```
SELECT ciudad FROM clientes EXCEPT
SELECT ciudad_dep FROM departamentos;
```

  - Usando IN

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

  - Usando EXISTS

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

Vistas

Vistas

- Una vista es una tabla ficticia(no existen como un conjunto de valores almacenados en la base de datos) que se construye a partir de una consulta a una tabla real. Para definir una vista se usa la siguiente sintaxis:

CREATE VIEW nombre_vista [(lista_columnas)] **AS** (consulta)
[**WITH CHECK OPTION**];

- donde se indica el nombre de la vista, a continuación se pueden especificar los nombres de las columnas de la vista, se define la consulta que construirá la vista, y se puede añadir la clausula "with check option" para evitar inserciones o actualizaciones excepto en los registros en que la cláusula WHERE de la consulta se evalúe como true.

Vistas

- Para borrar una vista se utiliza la sentencia **DROP VIEW**:

```
DROP VIEW nombre_vista (RESTRICT|CASCADE);
```

- donde:
  - 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.

Vistas

- Para ilustrar las vistas, se van a considerar las siguientes tablas:
  - 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 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

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_cli, num_pedidos) AS
(SELECT c.codigo_cli, COUNT(*) FROM pedidos p, clientes c WHERE p.codigo_cliente = c.codigo_cli GROUP BY c.codigo_cli);
```

- Y se obtendría la vista:

pedidos_por_clientes	
codigo_cli	num pedidos
10	2
20	1
30	1

Enlaces para saber más

Bibliografía

- "Fundamentos de Sistemas de Bases de Datos". Elmasri, Navathe. Editorial Pearson Educación. 5ª Edición.
- "Fundamentos de Bases de Datos". Silberschatz, Korth, Sudarshan. Editorial McGraw-Hill. 5ª Edición.
- "Introducción a las Bases de Datos". Ullman, Widom. Editorial Pearson Educación. 1999.
- "*Database systems: the complete book*". García Molina, H., Ullman, J., Widom, J.. 2nd ed. Pearson Education International, cop. 2009.
- "An Introduction to Database Systems". Date, C. J. Pearson Addison Wesley. 2004.

Enlaces

- Estándar SQL:

http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498

- Libro online:

http://en.wikibooks.org/wiki/Structured_Query_Language