Base de datos

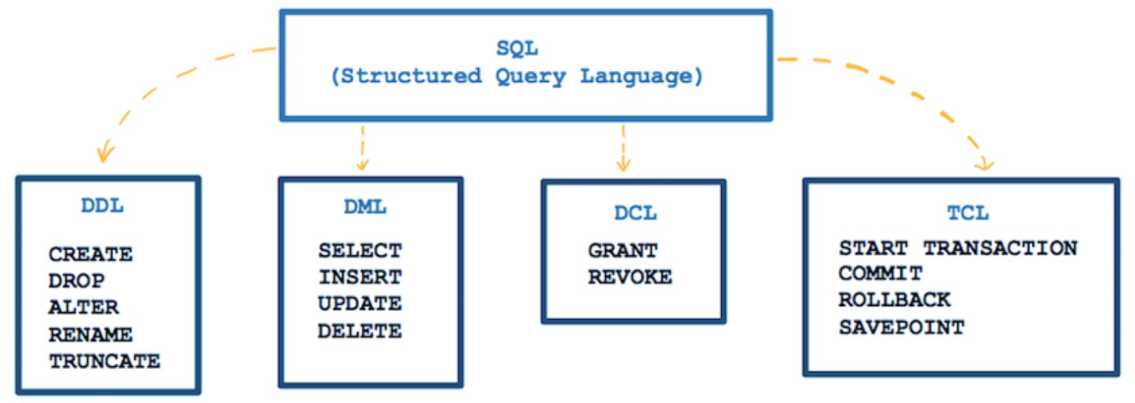
1º DAM

CURSO 2022-2023

Lenguaje de Manipulación de datos (DML)

CONSULTAS SQL SOBRE UNA TABLA - Realización de consultas SQL

El lenguaje DML de SQL



El DML (*Data Manipulation Language*) o Lenguaje de Manipulación de Datos es la parte de SQL dedicada a la manipulación de los datos. Las sentencias DML son las siguientes:

- SELECT: se utiliza para realizar consultas y extraer información de la base de datos.
- INSERT: se utiliza para insertar registros en las tablas de la base de datos.
- UPDATE: se utiliza para actualizar los registros de una tabla.
- DELETE: se utiliza para eliminar registros de una tabla.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Sintaxis de la instrucción SELECT

Según la documentación oficial de MySQL ésta sería la sintaxis para realizar una consulta con la sentencia SELECT en MySQL:

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
[SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references]
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING having_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_COUNT | row_COUNT OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
        [FOR UPDATE | LOCK IN SHARE MODE]]
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Sintaxis de la instrucción SELECT

Para empezar con consultas sencillas podemos simplificar la definición anterior y quedarnos con la siguiente:

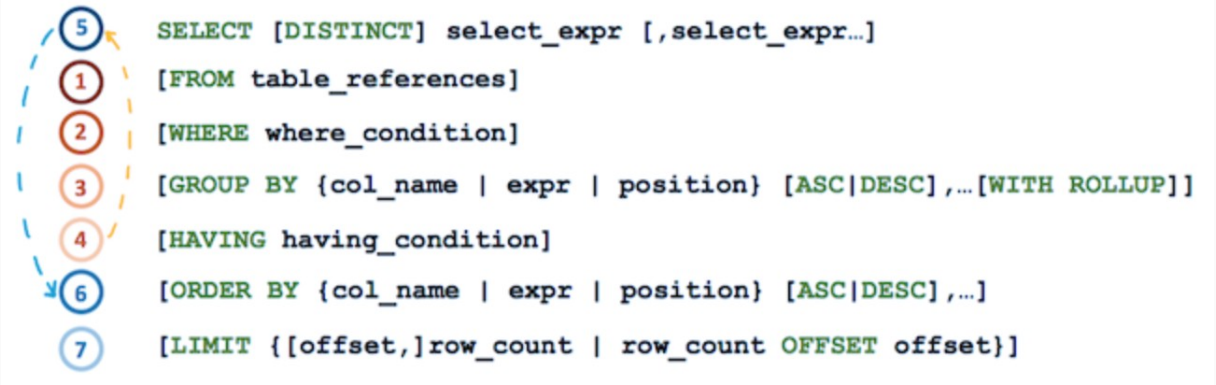
```
SELECT [DISTINCT] select_expr [, select_expr ...]
[FROM table_references]
[WHERE where_condition]
[GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
[HAVING having_condition]
[ORDER BY {col_name | expr | position} [ASC | DESC], ...]
[LIMIT {[offset,] row_COUNT | row_COUNT OFFSET offset}]
```

Es muy importante conocer en qué orden se ejecuta cada una de las cláusulas que forman la sentencia SELECT. El orden de ejecución es el siguiente:

- Cláusula FROM.
- Cláusula WHERE (Es opcional, puede ser que no aparezca).
- Cláusula GROUP BY (Es opcional, puede ser que no aparezca).
- Cláusula HAVING (Es opcional, puede ser que no aparezca).
- Cláusula SELECT.
- Cláusula ORDER BY (Es opcional, puede ser que no aparezca).
- Cláusula LIMIT (Es opcional, puede ser que no aparezca).

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Sintaxis de la instrucción SELECT



Hay que tener en cuenta que el resultado de una consulta siempre será una tabla de datos, que puede tener una o varias columnas y ninguna, una o varias filas.

El hecho de que el resultado de una consulta sea una tabla es muy interesante porque nos permite realizar cosas como almacenar los resultados como una nueva en la base de datos. También será posible combinar el resultado de dos o más consultas para crear una tabla mayor con la unión de los dos resultados. Además, los resultados de una consulta también pueden ser consultados por otras nuevas consultas.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Nos permite indicar cuáles serán las columnas que tendrá la tabla de resultados de la consulta que estamos realizando. Las opciones que podemos indicar son las siguientes:

- El nombre de una columna de la tabla sobre la que estamos realizando la consulta. Será una columna de la tabla que aparece en la cláusula FROM.
- Una constante que aparecerá en todas las filas de la tabla resultado.
- Una expresión que nos permite calcular nuevos valores.

CONSULTAS SQL SOBRE UNA TABLA

Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener los datos de todas las columnas de una tabla (SELECT *)

Ejemplo: Vamos a utilizar la siguiente base de datos de ejemplo para MySQL.

```
DROP DATABASE IF EXISTS instituto;
CREATE DATABASE instituto CHARACTER SET utf8mb4;
USE instituto;

CREATE TABLE alumno (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  apellido1 VARCHAR(100) NOT NULL,
  apellido2 VARCHAR(100),
  fecha_nacimiento DATE NOT NULL,
  es_repetidor ENUM('sí', 'no') NOT NULL,
  teléfono VARCHAR(9)
);

INSERT INTO alumno VALUES(1, 'María', 'Sánchez', 'Pérez', '1990/12/01', 'no', NULL);
INSERT INTO alumno VALUES(2, 'Juan', 'Sáez', 'Vega', '1998/04/02', 'no', 618253876);
INSERT INTO alumno VALUES(3, 'Pepe', 'Ramírez', 'Gea', '1988/01/03', 'no', NULL);
INSERT INTO alumno VALUES(4, 'Lucía', 'Sánchez', 'Ortega', '1993/06/13', 'sí', 678516294);
INSERT INTO alumno VALUES(5, 'Paco', 'Martínez', 'López', '1995/11/24', 'no', 692735409);
INSERT INTO alumno VALUES(6, 'Irene', 'Gutiérrez', 'Sánchez', '1991/03/28', 'sí', NULL);
INSERT INTO alumno VALUES(7, 'Cristina', 'Fernández', 'Ramírez', '1996/09/17', 'no',
628349590);
```

CONSULTAS SQL SOBRE UNA TABLA

Supongamos que tenemos una tabla llamada **alumno** con la siguiente información de los alumnos matriculados en un determinado curso.

Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener los datos de todas las columnas de una tabla (SELECT *)

id	nombre	apellido1	apellido2	fecha_nacimiento	es_repetidor	teléfono
1	María	Sánchez	Pérez	1990/12/01	no	NULL
2	Juan	Sáez	Vega	1998/04/02	no	618253876
3	Pepe	Ramírez	Gea	1988/01/03	no	NULL
4	Lucía	Sánchez	Ortega	1993/06/13	sí	678516294
5	Paco	Martínez	López	1995/11/24	no	692735409
6	Irene	Gutiérrez	Sánchez	1991/03/28	sí	NULL
7	Cristina	Fernández	Ramírez	1996/09/17	no	628349590
8	Antonio	Carretero	Ortega	1994/05/20	sí	612345633
9	Manuel	Domínguez	Hernández	1999/07/08	no	NULL
10	Daniel	Moreno	Ruiz	1998/02/03	no	NULL

Vamos a ver qué consultas sería necesario realizar para obtener los siguientes datos.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener los datos de todas las columnas de una tabla (SELECT *)

1. **Obtener todos los datos de todos los alumnos matriculados en el curso.**

```
SELECT *
FROM alumno;
```

El carácter * es un comodín que utilizamos para indicar que queremos seleccionar todas las columnas de la tabla. La consulta anterior devolverá todos los datos de la tabla.

Tenga en cuenta que las palabras reservadas de SQL no son *case sensitive*, por lo tanto es posible escribir la sentencia anterior de la siguiente forma obteniendo el mismo resultado:

```
select *
from alumno;
```

Otra consideración que también debemos tener en cuenta es que una consulta SQL se puede escribir en una o varias líneas. Por ejemplo, la siguiente sentencia tendría el mismo resultado que la anterior:

```
SELECT * FROM alumno;
```

Vamos a considerar como una buena práctica escribir las consultas SQL en varias líneas, empezando cada línea con la palabra reservada de la cláusula correspondiente que forma la consulta.

CONSULTAS SQL SOBRE UNA TABLA

Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener los datos de algunas columnas de una tabla

2. Obtener el nombre de todos los alumnos.

```
SELECT nombre
FROM alumno;
```

nombre
María
Juan
Pepe
Lucía
Paco
Irene
Cristina
Antonio
Manuel
Daniel

3. Obtener el nombre y los apellidos de todos los alumnos.

```
SELECT nombre, apellido1, apellido2
FROM alumno;
```

nombre	apellido1	apellido2
María	Sánchez	Pérez
Juan	Sáez	Vega
Pepe	Ramírez	Gea
Lucía	Sánchez	Ortega
Paco	Martínez	López
Irene	Gutiérrez	Sánchez
Cristina	Fernández	Ramírez
Antonio	Carretero	Ortega
Manuel	Domínguez	Hernández
Daniel	Moreno	Ruiz

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener los datos de algunas columnas de una tabla

3. Obtener el nombre y los apellidos de todos los alumnos.
Tenga en cuenta que el resultado de la consulta SQL mostrará las columnas que haya solicitado, siguiendo el orden en el que se hayan indicado. Por lo tanto la siguiente consulta:

```
SELECT apellido1, apellido2, nombre
FROM alumno;
```

Devolverá lo siguiente:

apellido1	apellido2	nombre
Sánchez	Pérez	María
Sáez	Vega	Juan
Ramírez	Gea	Pepe
Sánchez	Ortega	Lucía
Martínez	López	Paco
Gutiérrez	Sánchez	Irene
Fernández	Ramírez	Cristina
Carretero	Ortega	Antonio
Domínguez	Hernández	Manuel
Moreno	Ruiz	Daniel

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo realizar comentarios en sentencias SQL

Para escribir comentarios en nuestras sentencias SQL podemos hacerlo de diferentes formas.

```
-- Esto es un comentario
SELECT nombre, apellido1, apellido2 -- Esto es otro comentario
FROM alumno;
```

```
/* Esto es un comentario
   de varias líneas */
SELECT nombre, apellido1, apellido2 /* Esto es otro comentario */
FROM alumno;
```

```
# Esto es un comentario
SELECT nombre, apellido1, apellido2 # Esto es otro comentario
FROM alumno;
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener columnas calculadas

Ejemplo: Vamos a utilizar la siguiente base de datos de ejemplo para MySQL.

```
DROP DATABASE IF EXISTS tienda;
CREATE DATABASE tienda CHARACTER SET utf8mb4;
USE tienda;

CREATE TABLE ventas (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  cantidad_comprada INT UNSIGNED NOT NULL,
  precio_por_elemento DECIMAL(7,2) NOT NULL
);

INSERT INTO ventas VALUES (1, 2, 1.50);
INSERT INTO ventas VALUES (2, 5, 1.75);
INSERT INTO ventas VALUES (3, 7, 2.00);
INSERT INTO ventas VALUES (4, 9, 3.50);
INSERT INTO ventas VALUES (5, 6, 9.99);
```

Es posible realizar cálculos aritméticos entre columnas para calcular nuevos valores. Por ejemplo, supongamos que tenemos la siguiente tabla con información sobre ventas.

id	cantidad_comprada	precio_por_elemento
1	2	1.50
2	5	1.75
3	7	2.00
4	9	3.50
5	6	9.99

Y queremos calcular una nueva columna con el precio total de la venta, que sería equivalente a multiplicar el valor de la columna cantidad_comprada por precio_por_elemento.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener columnas calculadas

Para obtener esta nueva columna podríamos realizar la siguiente consulta:

```
SELECT id, cantidad_comprada, precio_por_elemento, cantidad_comprada * precio_por_elemento
FROM ventas;
```

Y el resultado sería el siguiente:

id	cantidad_comprada	precio_por_elemento	cantidad_comprada * precio_por_elemento
1	2	1.50	3.00
2	5	1.75	8.75
3	7	2.00	14.00
4	9	3.50	31.50
5	6	9.99	59.94

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener columnas calculadas

Ejemplo: Vamos a utilizar la siguiente base de datos de ejemplo para MySQL.

```
DROP DATABASE IF EXISTS company;
CREATE DATABASE company CHARACTER SET utf8mb4;
USE company;

CREATE TABLE offices (
  office INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  city VARCHAR(50) NOT NULL,
  region VARCHAR(50) NOT NULL,
  manager INT UNSIGNED,
  target DECIMAL(9,2) NOT NULL,
  sales DECIMAL(9,2) NOT NULL
);

INSERT INTO offices VALUES (11, 'New York', 'Eastern', 106,  575000, 692637);
INSERT INTO offices VALUES (12, 'Chicago', 'Eastern', 104, 800000, 735042);
INSERT INTO offices VALUES (13, 'Atlanta', 'Eastern', NULL, 350000, 367911);
INSERT INTO offices VALUES (21, 'Los Angeles', 'Western', 108, 725000, 835915);
INSERT INTO offices VALUES (22, 'Denver', 'Western', 108, 300000, 186042);
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo obtener columnas calculadas

Supongamos que tenemos una tabla llamada oficinas que contiene información sobre las ventas reales que ha generado y el valor de ventas esperado, y nos gustaría conocer si las oficinas han conseguido el objetivo propuesto, y si están por debajo o por encima del valor de ventas esperado.

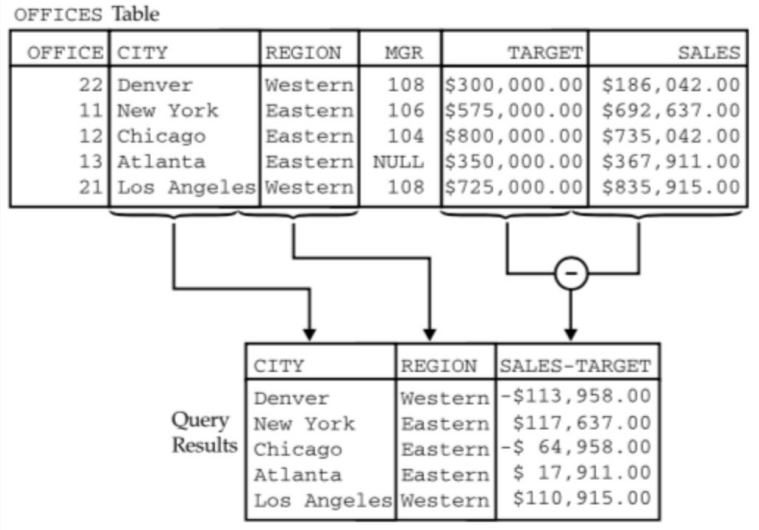


Imagen: Imagen extraída del libro *SQL: The Complete Reference* de James R. Groff y otros.

En este caso podríamos realizar la siguiente consulta:

```
SELECT city, region,  sales - target
FROM offices;
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo realizar alias de columnas con AS

Con la palabra reservada AS podemos crear alias para las columnas. Esto puede ser útil cuando estamos calculando nuevas columnas a partir de valores de las columnas actuales. En el ejemplo anterior de la tabla que contiene información sobre las ventas, podríamos crear el siguiente alias:

```
SELECT id, cantidad_comprada, precio_por_elemento, cantidad_comprada * precio_por_elemento AS 'precio total'
FROM ventas;
```

Si el nuevo nombre que estamos creando para el alias contiene espacios en blanco es necesario usar comillas simples. Al crear este alias obtendremos el siguiente resultado:

id	cantidad_comprada	precio_por_elemento	precio total
1	2	1.50	3.00
2	5	1.75	8.75
3	7	2.00	14.00

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo utilizar funciones de MySQL en la cláusula SELECT

Es posible hacer uso de funciones específicas de MySQL en la cláusula SELECT. MySQL nos ofrece funciones matemáticas, funciones para trabajar con cadenas y funciones para trabajar con fechas y horas. Algunos ejemplos de las funciones de MySQL que utilizaremos a lo largo del curso son las siguientes.

Funciones con cadenas

Función	Descripción
CONCAT	Concatena cadenas
CONCAT_WS	Concatena cadenas con un separador
LOWER	Devuelve una cadena en minúscula
UPPER	Devuelve una cadena en mayúscula
SUBSTR	Devuelve una subcadena

Funciones matemáticas

Función	Descripción
ABS()	Devuelve el valor absoluto
POW(x,y)	Devuelve el valor de x elevado a y
SQRT()	Devuelve la raíz cuadrada
PI()	Devuelve el valor del número PI
ROUND()	Redondea un valor numérico
TRUNCATE()	Trunca un valor numérico

Funciones de fecha y hora

Función	Descripción
NOW()	Devuelve la fecha y la hora actual
CURTIME()	Devuelve la hora actual

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo utilizar funciones de MySQL en la cláusula SELECT

Ejemplo: Obtener el nombre y los apellidos de todos los alumnos en una única columna.

```
SELECT CONCAT(nombre, apellido1, apellido2) AS nombre_completo
FROM alumno;
```

nombre_completo
MaríaSánchezPérez
JuanSáezVega
PepeRamírezGea
LucíaSánchezOrtega
PacoMartínezLópez
IreneGutiérrezSánchez
CristinaFernándezRamírez
AntonioCarreteroOrtega
ManuelDomínguezHernández
DanielMorenoRuiz

En este caso estamos haciendo uso de la función CONCAT de MySQL y la palabra reservada AS para crear un alias de la columna y renombrarla como *nombre_completo*.

La función CONCAT de MySQL no añade ningún espacio entre las columnas, por eso los valores de las tres columnas aparecen como una sola cadena sin espacios entre ellas. Para resolver este problema podemos hacer uso de la función CONCAT_WS que nos permite definir un carácter separador entre cada columna.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula SELECT

Cómo utilizar funciones de MySQL en la cláusula SELECT

Ejemplo: Obtener el nombre y los apellidos de todos los alumnos en una única columna.

En este ejemplo haremos uso de la función CONCAT_WS y usaremos un espacio en blanco como separador.

```
SELECT CONCAT_WS(' ', nombre, apellido1, apellido2) AS nombre_completo
FROM alumno;
```

nombre_completo
María Sánchez Pérez
Juan Sáez Vega
Pepe Ramírez Gea
Lucía Sánchez Ortega
Paco Martínez López
Irene Gutiérrez Sánchez
Cristina Fernández Ramírez
Antonio Carretero Ortega
Manuel Domínguez Hernández
Daniel Moreno Ruiz

Importante: La función CONCAT devolverá NULL cuando alguna de las cadenas que está concatenando es igual NULL, mientras que la función CONCAT_WS omitirá todas las cadenas que sean igual a NULL y realizará la concatenación con el resto de cadenas.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Modificadores ALL, DISTINCT y DISTINCTROW

Los modificadores ALL y DISTINCT indican si se deben incluir o no filas repetidas en el resultado de la consulta.

- ALL indica que se deben incluir todas las filas, incluidas las repetidas. Es la opción por defecto, por lo tanto no es necesario indicarla.
- DISTINCT elimina las filas repetidas en el resultado de la consulta.
- DISTINCTROW es un sinónimo de DISTINCT.

Ejemplo: En el siguiente ejemplo vamos a ver la diferencia que existe entre utilizar DISTINCT y no utilizarlo. La siguiente consulta mostrará todas las filas que existen en la columna apellido1 de la tabla alumno.

```
SELECT apellido1
FROM alumno;
```

apellido1
Sánchez
Sáez
Ramírez
Sánchez
Martínez
Gutiérrez
Fernández
Carretero
Domínguez
Moreno

Si en la consulta anterior utilizamos DISTINCT se eliminarán todos los valores repetidos que existan.

```
SELECT DISTINCT apellido1
FROM alumno;
```

apellido1
Sánchez
Sáez
Ramírez
Martínez
Gutiérrez
Fernández
Carretero
Domínguez
Moreno

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Modificadores ALL, DISTINCT y DISTINCTROW

Si en la cláusula SELECT utilizamos DISTINCT con más de una columna, la consulta seguirá eliminando todas las filas repetidas que existan. Por ejemplo, si tenemos las columnas apellido1, apellido2 y nombre, se eliminarán todas las filas que tengan los mismos valores en las tres columnas.

```
SELECT DISTINCT apellido1, apellido2, nombre
FROM alumno;
```

Apellido1	apellido2	nombre
Sánchez	Pérez	María
Sáez	Vega	Juan
Ramírez	Gea	Pepe
Sánchez	Ortega	Lucía
Martínez	López	Paco
Gutiérrez	Sánchez	Irene
Fernández	Ramírez	Cristina
Carretero	Ortega	Antonio
Domínguez	Hernández	Manuel
Moreno	Ruiz	Daniel

En este ejemplo no se ha eliminado ninguna fila porque no existen alumnos que tengan los mismos apellidos y el mismo nombre.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula ORDER BY

ORDER BY permite ordenar las filas que se incluyen en el resultado de la consulta. La sintaxis de MySQL es la siguiente:

```
[ORDER BY {col_name | expr | position} [ASC | DESC], ...]
```

Esta cláusula nos permite ordenar el resultado de forma ascendente ASC o descendente DESC, además de permitirnos ordenar por varias columnas estableciendo diferentes niveles de ordenación.

Cómo ordenar de forma ascendente

Ejemplo

1. Obtener el nombre y los apellidos de todos los alumnos, ordenados por su primer apellido de forma ascendente.

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY apellido1;
```

Si no indicamos nada en la cláusula ORDER BY se ordenará por defecto de forma ascendente.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula ORDER BY

Cómo ordenar de forma ascendente

La consulta anterior es equivalente a esta otra.

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY apellido1 ASC;
```

El resultado de ambas consultas será:

nombre	apellido1	apellido2
Carretero	Ortega	Antonio
Domínguez	Hernández	Manuel
Fernández	Ramírez	Cristina
Gutiérrez	Sánchez	Irene
Martínez	López	Paco
Moreno	Ruiz	Daniel
Ramírez	Gea	Pepe
Sáez	Vega	Juan
Sánchez	Pérez	María
Sánchez	Ortega	Lucía

Las filas están ordenadas correctamente por el primer apellido, pero todavía hay que resolver cómo ordenar el listado cuando existen varias filas donde coincide el valor del primer apellido. En este caso tenemos dos filas donde el primer apellido es Sánchez:

nombre	apellido1	apellido2
…	…	…
Sánchez	Pérez	María
Sánchez	Ortega	Lucía

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula ORDER BY

Cómo ordenar de forma descendente

Ejemplo

1. Obtener el nombre y los apellidos de todos los alumnos, ordenados por su primer apellido de forma descendente.

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY apellido1 DESC;
```

Cómo ordenar utilizando múltiples columnas

Ejemplo

En este ejemplo vamos obtener un listado de todos los alumnos ordenados por el primer apellido, segundo apellido y nombre, de forma ascendente.

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY apellido1, apellido2, nombre;
```

En lugar de indicar el nombre de las columnas en la cláusula ORDER BY podemos indicar sobre la posición donde aparecen en la cláusula SELECT, de modo que la consulta anterior sería equivalente a la siguiente:

```
SELECT apellido1, apellido2, nombre
FROM alumno
ORDER BY 1, 2, 3;
```

CONSULTAS SQL SOBRE UNA TABLA

Consultas básicas sobre una tabla

Cláusula `LIMIT`

LIMIT permite limitar el número de filas que se incluyen en el resultado de la consulta. La sintaxis de MySQL es la siguiente:

```
[LIMIT {[offset,] row_COUNT | row_COUNT OFFSET offset}]
```

donde row_COUNT es el número de filas que queremos obtener y offset el número de filas que nos saltamos antes de empezar a contar. Es decir, la primera fila que se obtiene como resultado es la que está situada en la posición offset + 1.

Ejemplo

```
DROP DATABASE IF EXISTS google;
CREATE DATABASE google CHARACTER SET utf8mb4;
USE google;

CREATE TABLE resultado (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  descripcion VARCHAR(200) NOT NULL,
  url VARCHAR(512) NOT NULL
);

INSERT INTO resultado VALUES (1, 'Resultado 1', 'Descripción 1', 'http://....');
INSERT INTO resultado VALUES (2, 'Resultado 2', 'Descripción 2', 'http://....');
INSERT INTO resultado VALUES (3, 'Resultado 3', 'Descripción 3', 'http://....');
INSERT INTO resultado VALUES (4, 'Resultado 4', 'Descripción 4', 'http://....');
INSERT INTO resultado VALUES (5, 'Resultado 5', 'Descripción 5', 'http://....');
INSERT INTO resultado VALUES (6, 'Resultado 6', 'Descripción 6', 'http://....');
INSERT INTO resultado VALUES (7, 'Resultado 7', 'Descripción 7', 'http://....');
INSERT INTO resultado VALUES (8, 'Resultado 8', 'Descripción 8', 'http://....');
INSERT INTO resultado VALUES (9, 'Resultado 9', 'Descripción 9', 'http://....');
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula `WHERE`

La cláusula WHERE nos permite añadir filtros a nuestras consultas para seleccionar sólo aquellas filas que cumplen una determinada condición. Estas condiciones se denominan predicados y el resultado de estas condiciones puede ser verdadero, falso o desconocido.

Una condición tendrá un resultado desconocido cuando alguno de los valores utilizados tiene el valor NULL.

Podemos diferenciar cinco tipos de condiciones que pueden aparecer en la cláusula WHERE:

- Condiciones para comparar valores o expresiones.
- Condiciones para comprobar si un valor está dentro de un rango de valores.
- Condiciones para comprobar si un valor está dentro de un conjunto de valores.
- Condiciones para comparar cadenas con patrones.
- Condiciones para comprobar si una columna tiene valores a NULL.

Los **operandos** usados en las condiciones pueden ser nombres de columnas, constantes o expresiones.
Los **operadores** que podemos usar en las condiciones pueden ser aritméticos, de comparación, lógicos, etc.

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula `WHERE`

Operadores disponibles en MySQL

Operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo

Operadores de comparación

Operador	Descripción
<	Menor que
<=	Menor o igual
>	Mayor que
>=	Mayor o igual
<>	Distinto
!=	Distinto
=	Igual que

Operadores lógicos

Operador	Descripción
AND	Y lógica
&&	Y lógica
OR	O lógica
||	O lógica
NOT	Negación lógica
!	Negación lógica

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula `WHERE`

Operador BETWEEN

Sintaxis:

```
expresión [NOT] BETWEEN cota_inferior AND cota_superior
```

Se utiliza para comprobar si un valor está dentro de un rango de valores. Por ejemplo, si queremos obtener los pedidos que se han realizado durante el mes de enero de 2018 podemos realizar la siguiente consulta:

```
SELECT *
FROM pedido
WHERE fecha_pedido BETWEEN '2018-01-01' AND '2018-01-31';
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula `WHERE`

Operador IN

Este operador nos permite comprobar si el valor de una determinada columna está incluido en una lista de valores.

Ejemplo: Obtener todos los datos de los alumnos que tengan como primer apellido Sánchez, Martínez o Domínguez.

```
SELECT *
FROM alumno
WHERE apellido1 IN (`Sánchez`, `Martínez`, `Domínguez`);
```

Ejemplo: Obtener todos los datos de los alumnos que no tengan como primer apellido Sánchez, Martínez o Domínguez.

```
SELECT *
FROM alumno
WHERE apellido1 NOT IN (`Sánchez`, `Martínez`, `Domínguez`);
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula `WHERE`

Operador LIKE

Sintaxis:

```
columna [NOT] LIKE patrón
```

Se utiliza para comparar si una cadena de caracteres coincide con un patrón. En el patrón podemos utilizar cualquier carácter alfanumérico, pero hay dos caracteres que tienen un significado especial, el símbolo del porcentaje (%) y el carácter de subrayado (_).

- %: Este carácter equivale a cualquier conjunto de caracteres.
- _: Este carácter equivale a cualquier carácter (¡Ojo! pero sólo un carácter).

Ejemplos:

Devuelva un listado de todos los alumnos que su primer apellido empiece por la letra S.

```
SELECT *
FROM alumno
WHERE apellido1 LIKE 'S%';
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula `WHERE` Operador LIKE

Ejemplos:

Devuelva un listado de todos los alumnos que su primer apellido termine por la letra z.

```
SELECT *
FROM alumno
WHERE apellido1 LIKE '%z';
```

Devuelva un listado de todos los alumnos que su nombre tenga el carácter a.

```
SELECT *
FROM alumno
WHERE nombre LIKE '%a%';
```

Devuelva un listado de todos los alumnos que tengan un nombre de cuatro caracteres.

```
SELECT *
FROM alumno
WHERE nombre LIKE '____';
```

Devuelve un listado de todos los productos cuyo nombre empieza con estas cuatro letras ‘A%BC’.

En este caso, el patrón que queremos buscar contiene el carácter %, por lo tanto, tenemos que usar un carácter de escape.

```
SELECT *
FROM producto
WHERE nombre LIKE 'A$%BC%' ESCAPE '$';
```

Por defecto, se utiliza el carácter '' como carácter de escape. De modo que podríamos escribir la consulta de la siguiente manera.

```
SELECT *
FROM producto
WHERE nombre LIKE 'A\%BC%';
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Cláusula WHERE

Operador REGEXP y expresiones regulares

El operador REGEXP nos permite realizar búsquedas mucho más potentes haciendo uso de expresiones regulares. Puede consultar la documentación oficial para conocer más detalles sobre cómo usar este operador.

Operador IS e IS NOT

Estos operadores nos permiten comprobar si el valor de una determinada columna es NULL o no lo es.

Ejemplo: Obtener la lista de alumnos que tienen un valor NULL en la columna teléfono.

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

Ejemplo: Obtener la lista de alumnos que no tienen un valor NULL en la columna teléfono.

```
SELECT *
FROM alumno
WHERE teléfono IS NOT NULL;
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Funciones disponibles en MySQL

Funciones con cadenas

A continuación se muestran algunas funciones disponibles en MySQL que pueden ser utilizadas para realizar consultas.

Las funciones se pueden utilizar en las cláusulas SELECT, WHERE y ORDER BY

Función	Descripción
CONCAT	Concatena cadenas
CONCAT_WS	Concatena cadenas con un separador
LOWER	Devuelve una cadena en minúscula
UPPER	Devuelve una cadena en mayúscula
SUBSTR	Devuelve una subcadena
LEFT	Devuelve los caracteres de una cadena, empezando por la izquierda
RIGHT	Devuelve los caracteres de una cadena, empezando por la derecha
CHAR_LENGTH	Devuelve el número de caracteres que tiene una cadena
LENGTH	Devuelve el número de bytes que ocupa una cadena
REVERSE	Devuelve una cadena invirtiendo el orden de sus caracteres
LTRIM	Elimina los espacios en blanco que existan al inicio de una cadena
RTRIM	Elimina los espacios en blanco que existan al final de una cadena
TRIM	Elimina los espacios en blanco que existan al inicio y al final de una cadena
REPLACE	Permite reemplazar un carácter dentro de una cadena

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Funciones disponibles en MySQL Funciones de fecha y hora

Función	Descripción
NOW()	Devuelve la fecha y la hora actual
CURTIME()	Devuelve la hora actual
ADDDATE	Suma un número de días a una fecha y calcula la nueva fecha
DATE_FORMAT	Nos permite formatear fechas
DATEDIFF	Calcula el número de días que hay entre dos fechas
YEAR	Devuelve el año de una fecha
MONTH	Devuelve el mes de una fecha
MONTHNAME	Devuelve el nombre del mes de una fecha
DAY	Devuelve el día de una fecha
DAYNAME	Devuelve el nombre del día de una fecha
HOUR	Devuelve las horas de un valor de tipo DATETIME
MINUTE	Devuelve los minutos de un valor de tipo DATETIME
SECOND	Devuelve los segundos de un valor de tipo DATETIME

Configuración regional en MySQL Server (*locale*)

Importante: Tenga en cuenta que para que los nombres de los meses y las abreviaciones aparezcan en español deberá configurar la variable del sistema lc_time_names. Esta variable afecta al resultado de las funciones DATE_FORMAT, DAYNAME y MONTHNAME.

En MySQL las variables se pueden definir como variables globales o variables de sesión. La diferencia que existe entre ellas es que una variable de sesión pierde su contenido cuando cerramos la sesión con el servidor, mientras que una variable global mantiene su valor hasta que se realiza un reinicio del servicio o se modifica por otro valor. Las variables globales sólo pueden ser configuradas por usuarios con privilegios de administración.

CONSULTAS SQL SOBRE UNA TABLA

Consultas básicas sobre una tabla

Funciones disponibles en MySQL

Funciones de fecha y hora

Configuración regional en MySQL Server (*locale*)

Para configurar la variable lc_time_names como una variable global, con la configuración regional de España tendrá que utilizar la palabra reservada GLOBAL, como se indica en el siguiente ejemplo.

```
SET GLOBAL lc_time_names = 'es_ES';
```

Para realizar la configuración como una variable de sesión tendría que ejecutar:

```
SET lc_time_names = 'es_ES';
```

Una vez hecho esto podrá consultar sus valores haciendo:

```
SELECT @@GLOBAL.lc_time_names, @@SESSION.lc_time_names;
```

En la documentación oficial pueden encontrar más información sobre la configuración regional en MySQL Server

Para consultar el valor de todas las variables de sesión del sistema podemos utilizar la sentencia:

```
SHOW VARIABLES;
```

O con el modificador SESSION:

```
SHOW SESSION VARIABLES;
```

Para consultar el valor de todas las variables globales del sistema podemos utilizar la sentencia:

```
SHOW GLOBAL VARIABLES;
```

CONSULTAS SQL SOBRE UNA TABLA

Consultas básicas sobre una tabla

Funciones disponibles en MySQL

Funciones de fecha y hora

Configuración de la zona horaria en MySQL Server (*time zone*)

Importante: Tenga en cuenta que también será necesario configurar nuestra zona horaria para que las funciones relacionadas con la hora devuelvan los valores de nuestra zona horaria. En este caso tendrá que configurar la variable del sistema time_zone, como variable global o como variable de sesión. A continuación, se muestra un ejemplo de cómo habría que configurar las variables para la zona horaria de Madrid.

```
SET GLOBAL time_zone = 'Europe/Madrid';
SET time_zone = 'Europe/Madrid';
```

Podemos comprobar el estado de las variables haciendo:

```
SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

En la documentación oficial pueden encontrar más información sobre la configuración de la zona horaria en MySQL Server

Ejemplos:

NOW() devuelve la fecha y la hora actual.

```
SELECT NOW();
```

CURTIME() devuelve la hora actual.

```
SELECT CURTIME();
```

CONSULTAS SQL SOBRE UNA TABLA
Consultas básicas sobre una tabla

Funciones disponibles en MySQL

Funciones matemáticas

Función	Descripción
ABS()	Devuelve el valor absoluto
POW(x,y)	Devuelve el valor de x elevado a y
SQRT	Devuelve la raíz cuadrada
PI()	Devuelve el valor del número PI
ROUND	Redondea un valor numérico
TRUNCATE	Trunca un valor numérico
CEIL	Devuelve el entero inmediatamente superior o igual
FLOOR	Devuelve el entero inmediatamente inferior o igual
MOD	Devuelve el resto de una división

En la documentación oficial puede encontrar la referencia completa de todas las funciones matemáticas disponibles en MySQL

Ejemplos:

ABS devuelve el valor absoluto de un número.

```
SELECT ABS(-25);
  25
```

POW(x, y) devuelve el valor de x elevado a y.

```
SELECT POW(2, 10);
  1024
```

SQRT devuelve la raíz cuadrada de un número.

```
SELECT SQRT(1024);
  32
```

PI() devuelve el valor del número PI.

```
SELECT PI();
  3.141593
```

ROUND redondea un valor numérico

```
SELECT ROUND(37.6234);
  38
```

TRUNCATE Trunca un valor numérico.

```
SELECT TRUNCATE(37.6234, 0);
  37
```

CEIL devuelve el entero inmediatamente superior o igual al parámetro de entrada.

```
SELECT CEIL(4.3);
  5
```

FLOOR devuelve el entero inmediatamente inferior o igual al parámetro de entrada.

```
SELECT FLOOR(4.3);
  4
```

CONSULTAS SQL SOBRE UNA TABLA - Consultas básicas sobre una tabla

Funciones disponibles en MySQL

Funciones para realizar búsquedas de tipo FULLTEXT

Ejemplo

En este ejemplo vamos a crear un índice FULLTEXT sobre las columnas nombre y descripción de la tabla producto, para permitir realizar búsquedas más eficientes sobre esas columnas.

```
CREATE FULLTEXT INDEX idx_nombre_descripcion ON producto(nombre, descripcion);
```

Una vez creado el índice ejecutamos la consulta haciendo uso de MATCH y AGAINST.

```
SELECT *
FROM producto
WHERE MATCH(nombre, descripcion) AGAINST ('acero');
```

Puedes encontrar más información sobre la creación de índices en MySQL en la documentación oficial.

CONSULTAS SQL SOBRE UNA TABLA

Errores comunes

Error al comprobar si una columna es NULL

Ejemplo: Obtener la lista de alumnos que tienen un valor NULL en la columna teléfono.

Consulta incorrecta.

```
SELECT *
FROM alumno
WHERE teléfono = NULL;
```

Consulta correcta.

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

Error al comparar cadenas con patrones utilizando el operador =

Ejemplo: Devuelve un listado de los alumnos cuyo primer apellido empieza por S.

Consulta incorrecta.

```
SELECT *
FROM alumno
WHERE apellido1 = 'S%';
```

Consulta correcta.

```
SELECT *
FROM alumno
WHERE apellido1 LIKE 'S%';
```

CONSULTAS SQL SOBRE UNA TABLA

Errores comunes

Error al comparar cadenas con patrones utilizando el operador !=

Ejemplo: Devuelve un listado de los alumnos cuyo primer apellido no empieza por S.

Consulta incorrecta.

```
SELECT *
FROM alumno
WHERE apellido1 != 'S%';
```

Consulta correcta.

```
SELECT *
FROM alumno
WHERE apellido1 NOT LIKE 'S%';
```

Error al comparar un rango de valores con AND

Cuando queremos comparar si un valor está dentro de un rango tenemos que utilizar dos condiciones unidas con la operación lógica AND. En el siguiente ejemplo se muestra una consulta incorrecta donde la segunda condición siempre será verdadera porque no estamos comparando con ningún valor, estamos poniendo un valor constante que al ser distinto de 0 siempre nos dará un valor verdadero como resultado de la comparación.

Consulta incorrecta

```
SELECT *
FROM alumno
WHERE fecha_nacimiento >= '1999/01/01' AND '1999/12/31'
```

Consulta correcta

```
SELECT *
FROM alumno
WHERE fecha_nacimiento >= '1999/01/01' AND fecha_nacimiento <= '1999/12/31'
```

CONSULTAS SQL SOBRE UNA TABLA - Errores comunes

Errores de conversión de tipos en la evaluación de expresiones

Cuando se utiliza un operador con operandos de diferentes tipos de datos, MySQL realiza una conversión automática de tipo para que los operandos sean compatibles. Por ejemplo, convierte automáticamente cadenas en números según sea necesario y viceversa.

```
SELECT 1 + '1';
    -> 2
```

```
SELECT CONCAT(1, '1');
    -> '11'
```

Ejemplo: Resta entre dos valores de tipo VARCHAR

Las cadenas las convierte a datos de tipo entero de la mejor forma posible y luego realiza la resta. En este caso la cadena '2021-01-31' la convierte en el número entero 2021 y la cadena '2021-02-01' en el número entero 2021, por este motivo el resultado de la resta es 0.

```
SELECT '2021-01-31' - '2021-02-01';
    -> 0
```

Ejemplo: Resta entre dos valores de tipo DATE

En este ejemplo estamos convirtiendo las cadenas a un valor de tipo DATE y luego se realiza la resta. En este caso, los datos de tipo DATE los convierte a datos de tipo entero pero de una forma más precisa que la conversión anterior, ya que la cadena '2008-08-31' la convierte en el número entero 20080831 y la cadena '2008-09-01' en el número entero 20080901, por este motivo el resultado de la resta es -70.

```
SELECT CAST('2021-01-31' AS DATE) - CAST('2021-02-01' AS DATE);
    -> -70
```

CONSULTAS SQL SOBRE UNA TABLA - Errores comunes

Errores de conversión de tipos en la evaluación de expresiones

Ejemplo: Resta entre dos valores de tipo INT

```
SELECT 20210131 - 20210201;
    -> -70
```

Ejemplo: Resta entre dos valores de tipo DATE utilizando la función DATEDIFF

```
SELECT DATEDIFF('2021-01-31', '2021-02-01');
    -> -1
```

Puede encontrar más información sobre la conversión de tipos en la evaluación de expresiones en la documentación oficial.

CONSULTAS SQL SOBRE UNA TABLA - Errores comunes

Error al utilizar los operadores de suma y resta entre datos de tipo `DATE`, `DATETIME` y `TIMESTAMP`

Para poder realizar operaciones de suma y resta entre datos de tipo DATE, DATETIME y TIMESTAMP es necesario hacer uso de las funciones específicas de fecha y hora disponibles en MySQL. Si tratamos de realizar una operación de suma o resta entre valores de tipo DATE, DATETIME y TIMESTAMP podemos obtener un resultado incorrecto en algunas situaciones, ya que estos valores se convierten a un dato de tipo numérico y posteriormente se realiza la operación entre ellos.

Consulta incorrecta

```
SELECT fecha_esperada, fecha_entrega,  fecha_entrega - fecha_esperada
FROM pedido;
```

fecha_esperada	fecha_entrega	fecha_entrega - fecha_esperada
2021-01-31	2021-02-01	-70

CONSULTAS SQL SOBRE UNA TABLA - Errores comunes

Error al utilizar los operadores de suma y resta entre datos de tipo `DATE`, `DATETIME` y `TIMESTAMP`

En este ejemplo estamos realizando una resta entre dos datos de tipo DATE utilizando el operador -. Lo que ocurre en este caso, es que antes de realizar la operación de resta, MySQL convierte los datos de tipo DATE a datos de tipo entero. De forma que la fecha 2008-08-31 la convierte en el número entero 20080831 y la fecha 2008-09-01 en el número entero 20080901. Por este motivo el resultado que obtenemos a realizar la resta es -70.

Consulta correcta

```
SELECT fecha_esperada, fecha_entrega,  DATEDIFF(fecha_entrega, fecha_esperada)
FROM pedido;
```

fecha_esperada	fecha_entrega	DATEDIFF(fecha_entrega, fecha_esperada)
2021-01-31	2021-02-01	-1

En este caso se está utilizando la función DATEDIFF en lugar del operador de resta -. Al ser una función específica para trabajar con este tipo de dato obtenemos el resultado esperado.

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Las consultas multitabla nos permiten consultar información en más de una tabla. La única diferencia respecto a las consultas sencillas es que vamos a tener que especificar en la cláusula FROM cuáles son las tablas que vamos a usar y cómo las vamos a relacionar entre sí.

Para realizar este tipo de consultas podemos usar dos alternativas, la sintaxis de SQL 1 (SQL-86), que consiste en realizar el producto cartesiano de las tablas y añadir un filtro para relacionar los datos que tienen en común, y la sintaxis de SQL 2 (SQL-92 y SQL-2003) que incluye todas las cláusulas de tipo JOIN.

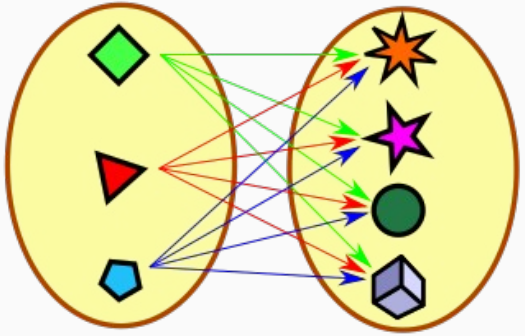
CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 1

Composiciones cruzadas (Producto cartesiano)

El **producto cartesiano** de dos conjuntos, es una operación que consiste en obtener otro conjunto cuyos elementos son **todas las parejas que pueden formarse entre los dos conjuntos**. Por ejemplo, tendríamos que coger el primer elemento del primer conjunto y formar una pareja con cada uno de los elementos del segundo conjunto. Una vez hecho esto, repetimos el mismo proceso para cada uno de los elementos del primer conjunto.



CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 1 - Composiciones cruzadas (Producto cartesiano)

Ejemplo: Suponemos que tenemos una base de datos con dos tablas: empleado y departamento.

```
SELECT *
FROM empleado;
```

codigo	nif	nombre	apellido1	apellido2	codigo_departamento
1	32481596F	Aarón	Rivero	Gómez	1
2	Y5575632D	Adela	Salas	Díaz	2
3	R6970642B	Adolfo	Rubio	Flores	3

```
SELECT *
FROM departamento;
```

codigo	nombre	presupuesto
1	Desarrollo	120000
2	Sistemas	150000
3	Recursos Humanos	280000

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 1 - Composiciones cruzadas (Producto cartesiano)

Ejemplo: Suponemos que tenemos una base de datos con dos tablas: empleado y departamento.

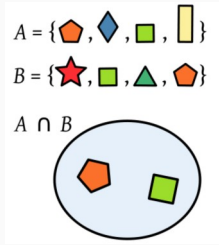
El producto cartesiano de las dos tablas se realiza con la siguiente consulta:

```
SELECT *
FROM empleado, departamento;
```

El resultado sería el siguiente:

codigo	nif	nombre	apellido1	apellido2	codigo_departamento	codigo	nombre	presupuesto	gastos
1	32481596F	Aarón	Rivero	Gómez	1	1	Desarrollo	120000	6000
2	Y5575632D	Adela	Salas	Díaz	2	1	Desarrollo	120000	6000
3	R6970642B	Adolfo	Rubio	Flores	3	1	Desarrollo	120000	6000
1	32481596F	Aarón	Rivero	Gómez	1	2	Sistemas	150000	21000
2	Y5575632D	Adela	Salas	Díaz	2	2	Sistemas	150000	21000
3	R6970642B	Adolfo	Rubio	Flores	3	2	Sistemas	150000	21000
1	32481596F	Aarón	Rivero	Gómez	1	3	Recursos Humanos	280000	25000

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA



Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 1 - Composiciones internas (Intersección)

La **intersección de dos conjuntos** es una operación que resulta en otro conjunto que contiene **sólo los elementos comunes** que existen en ambos conjuntos.

Ejemplo: Para poder realizar una operación de intersección entre las dos tablas debemos utilizar la cláusula WHERE para indicar la columna con la que queremos relacionar las dos tablas. Por ejemplo, para obtener un listado de los empleados y el departamento donde trabaja cada uno podemos realizar la siguiente consulta:

```
SELECT *
FROM empleado, departamento
WHERE empleado.codigo_departamento = departamento.codigo
```

El resultado sería el siguiente:

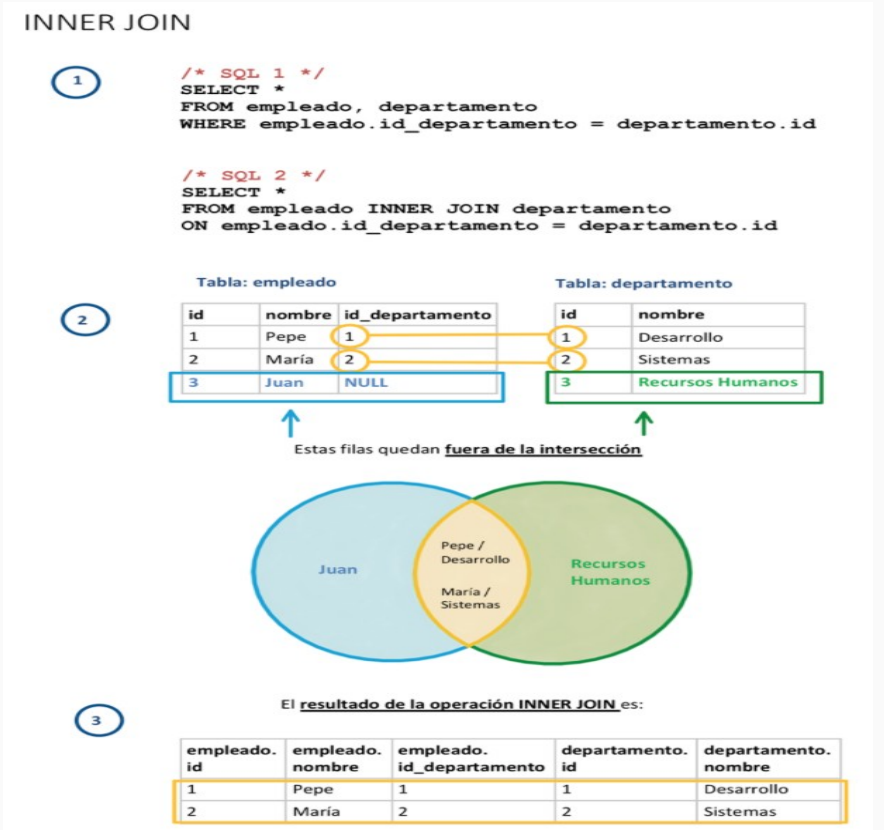
codigo	nif	nombre	apellido1	apellido2	codigo_departamento	codigo	nombre	presupuesto	gastos
1	32481596F	Aarón	Rivero	Gómez	1	1	Desarrollo	120000	6000
2	Y5575632D	Adela	Salas	Díaz	2	2	Sistemas	150000	21000
3	R6970642B	Adolfo	Rubio	Flores	3	3	Recursos Humanos	280000	25000

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 1 - Composiciones internas (Intersección)

Nota: Tenga en cuenta que con la **operación de intersección** sólo obtendremos los elementos de existan en ambos conjuntos. Por lo tanto, en el ejemplo anterior puede ser que existan filas en la tabla empleado que no aparecen en el resultado porque no tienen ningún departamento asociado, al igual que pueden existir filas en la tabla departamento que no aparecen en el resultado porque no tienen ningún empleado asociado.



CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 2

Composiciones cruzadas

- Producto cartesiano
  - CROSS JOIN

Ejemplo de CROSS JOIN:

```
SELECT *
FROM empleado CROSS JOIN departamento
```

Esta consulta nos devolvería el producto cartesiano de las dos tablas.

Composiciones internas

- Join interna
  - INNER JOIN o JOIN
  - NATURAL JOIN

Ejemplo de INNER JOIN:

```
SELECT *
FROM empleado INNER JOIN departamento
ON empleado.codigo_departamento = departamento.codigo
```

Esta consulta nos devolvería la intersección entre las dos tablas.

La palabra reservada INNER es opcional, de modo que la consulta anterior también se puede escribir así:

```
SELECT *
FROM empleado JOIN departamento
ON empleado.codigo_departamento = departamento.codigo
```

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 2 - Composiciones internas

NOTA: Tenga en cuenta que **si olvidamos incluir la cláusula ON obtendremos el producto cartesiano de las dos tablas**.

Por ejemplo, la siguiente consulta nos devolverá el producto cartesiano de las tablas empleado y departamento.

```
SELECT *
FROM empleado INNER JOIN departamento
```

Ejemplo de NATURAL JOIN:

```
SELECT *
FROM empleado NATURAL JOIN departamento
```

Esta consulta nos devolvería la intersección de las dos tablas, pero utilizaría las columnas que tengan el mismo nombre para relacionarlas. En este caso usaría las columnas código y nombre. Sólo deberíamos utilizar una composición de tipo NATURAL JOIN cuando estemos seguros que los nombres de las columnas sobre las que quiero relacionar las dos tablas se llaman igual en las dos tablas. Lo normal es que no suelan tener el mismo nombre y que debamos usar una composición de tipo INNER JOIN.

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 2 - Composiciones externas

- Join externa
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN (No implementada en MySQL)
  - NATURAL LEFT OUTER JOIN
  - NATURAL RIGHT OUTER JOIN

Ejemplo de LEFT OUTER JOIN:

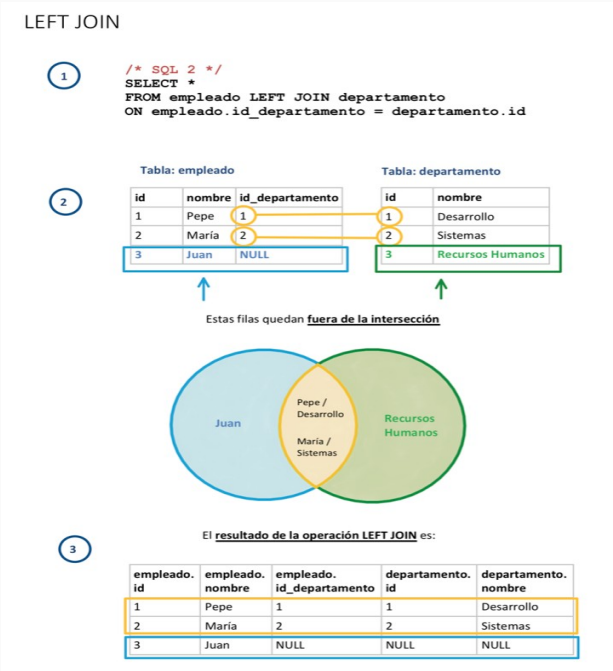
```
SELECT *
FROM empleado LEFT JOIN departamento
ON empleado.codigo_departamento = departamento.codigo
```

Esta consulta devolverá todas las filas de la tabla que hemos colocado a la izquierda de la composición, en este caso la tabla empleado. Y relacionará las filas de la tabla de la izquierda (empleado) con las filas de la tabla de la derecha (departamento) con las que encuentre una coincidencia. Si no encuentra ninguna coincidencia, se mostrarán los valores de la fila de la tabla izquierda (empleado) y en los valores de la tabla derecha (departamento) donde no ha encontrado una coincidencia mostrará el valor NULL.

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 2 - Composiciones externas



Ejemplo de RIGHT OUTER JOIN:

```
SELECT *
FROM empleado RIGHT JOIN departamento
ON empleado.codigo_departamento = departamento.codigo
```

Esta consulta devolverá todas las filas de la tabla que hemos colocado a la derecha de la composición, en este caso la tabla departamento. Y relacionará las filas de la tabla de la derecha (departamento) con las filas de la tabla de la izquierda (empleado) con las que encuentre una coincidencia. Si no encuentra ninguna coincidencia, se mostrarán los valores de la fila de la tabla derecha (departamento) y en los valores de la tabla izquierda (empleado) donde no ha encontrado una coincidencia mostrará el valor NULL.

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 2 - Composiciones externas

RIGHT JOIN

1

```
/* SQL 2 */
SELECT *
FROM empleado RIGHT JOIN departamento
ON empleado.id_departamento = departamento.id
```

2

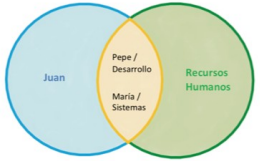
Tabla: empleado

id	nombre	id_departamento
1	Pepe	1
2	María	2
3	Juan	NULL

Tabla: departamento

id	nombre
1	Desarrollo
2	Sistemas
3	Recursos Humanos

Estas filas quedan **fuera de la intersección**



3

El **resultado de la operación RIGHT JOIN** es:

empleado.id	empleado.nombre	empleado.id_departamento	departamento.id	departamento.nombre
1	Pepe	1	1	Desarrollo
2	María	2	2	Sistemas
NULL	NULL	NULL	3	Recursos Humanos

Ejemplo de FULL OUTER JOIN:

La composición FULL OUTER JOIN no está implementada en MySQL, por lo tanto para poder simular esta operación será necesario hacer uso del operador UNION, que nos realiza la unión del resultado de dos consultas.

El resultado esperado de una composición de tipo FULL OUTER JOIN es obtener la intersección de las dos tablas, junto las filas de ambas tablas que no se puedan combinar. Dicho con otras palabras, el resultado sería el equivalente a realizar la unión de una consulta de tipo LEFT JOIN y una consultas de tipo RIGHT JOIN sobre las mismas tablas.

```
SELECT *
FROM empleado LEFT JOIN departamento
ON empleado.codigo_departamento = departamento.codigo

UNION

SELECT
FROM empleado RIGHT JOIN departamento
ON empleado.codigo_departamento = departamento.codigo
```

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 2 - Composiciones externas

Ejemplo de FULL OUTER JOIN:

Para poder utilizar el operador UNION entre dos o más consultas deberá tener en cuenta que:

- Deben tener el mismo número de columnas.
- Las columnas que se van a unir tienen que tener tipos de datos similares.

Para ordenar los resultados tras aplicar una operación de UNION existen dos soluciones:

- Usar la posición de la columna sobre la que queremos ordenar los resultados en el ORDER BY.
- Crear un alias en las columnas del primer SELECT sobre la que queremos ordenar los resultados y usarlo en el ORDER BY.

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 2 - Composiciones externas

Ejemplo de FULL OUTER JOIN:

```
/* Solución 1 */
SELECT departamento.nombre, empleado.apellido1, empleado.apellido2, empleado.nombre
FROM empleado LEFT JOIN departamento
ON empleado.codigo_departamento=departamento.codigo

UNION

SELECT departamento.nombre, empleado.apellido1, empleado.apellido2, empleado.nombre
FROM empleado RIGHT JOIN departamento
ON empleado.codigo_departamento=departamento.codigo

/* Solución 2 */
SELECT   departamento.nombre AS nombre_departamento, empleado.apellido1, empleado.apellido2,
empleado.nombre
FROM empleado LEFT JOIN departamento
ON empleado.codigo_departamento=departamento.codigo

UNION

SELECT departamento.nombre, empleado.apellido1, empleado.apellido2, empleado.nombre
FROM empleado RIGHT JOIN departamento
ON empleado.codigo_departamento=departamento.codigo

ORDER BY nombre_departamento;
```

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Consultas multitabla SQL 2 - Composiciones externas

Ejemplo de NATURAL LEFT JOIN:

```
SELECT *
FROM empleado NATURAL LEFT JOIN departamento
```

Esta consulta realiza un LEFT JOIN entre las dos tablas, la única diferencia es que en este caso no es necesario utilizar la cláusula ON para indicar sobre qué columna vamos a relacionar las dos tablas. **En este caso las tablas se van a relacionar sobre aquellas columnas que tengan el mismo nombre**. Por lo tanto, sólo deberíamos utilizar una composición de tipo NATURAL LEFT JOIN cuando estemos seguros de que los nombres de las columnas sobre las que quiero relacionar las dos tablas se llaman igual en las dos tablas.

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

El orden en las tablas no afecta al resultado final

Estas dos consultas devuelven el mismo resultado:

```
SELECT *
FROM empleado INNER JOIN departamento
ON empleado.codigo_departamento = departamento.codigo
```

```
SELECT *
FROM departamento INNER JOIN empleado
ON empleado.codigo_departamento = departamento.codigo
```

Podemos usar alias en las tablas

```
SELECT *
FROM empleado AS e INNER JOIN departamento AS d
ON e.codigo_departamento = d.codigo
```

```
SELECT *
FROM empleado e INNER JOIN departamento d
ON e.codigo_departamento = d.codigo
```

Unir tres o más tablas

Ejemplo:

```
SELECT *
FROM cliente INNER JOIN empleado
ON cliente.codigo_empleado_rep_ventas = empleado.codigo_empleado
INNER JOIN pago
ON cliente.codigo_cliente = pago.codigo_cliente;
```

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Composición interna y cruzada

Unir una tabla consigo misma (*self-equi-join*)

Para poder hacer una operación de INNER JOIN sobre la misma tabla es necesario utilizar un alias para la tabla. A continuación se muestra un ejemplo de las dos formas posibles de hacer una operación de INNER JOIN sobre la misma tablas haciendo uso de alias.

Ejemplo:

```
SELECT empleado.nombre, empleado.apellido1, empleado.apellido2, jefe.nombre, jefe.apellido1, jefe.apellido2
FROM empleado INNER JOIN empleado AS jefe
ON empleado.codigo_jefe = jefe.codigo_empleado
```

```
SELECT empleado.nombre, empleado.apellido1, empleado.apellido2, jefe.nombre, jefe.apellido1, jefe.apellido2
FROM empleado INNER JOIN empleado jefe
ON empleado.codigo_jefe = jefe.codigo_empleado
```

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Errores comunes

1. Nos olvidamos de incluir en el WHERE la condición que nos relaciona las dos tablas.

Consulta incorrecta

```
SELECT *
FROM producto, fabricante
WHERE fabricante.nombre = 'Lenovo';
```

Consulta correcta

```
SELECT *
FROM producto, fabricante
WHERE producto.codigo_fabricante = fabricante.codigo AND fabricante.nombre = 'Lenovo';
```

2. Nos olvidamos de incluir ON en las consultas de tipo INNER JOIN.

Consulta incorrecta

```
SELECT *
FROM producto INNER JOIN fabricante
WHERE fabricante.nombre = 'Lenovo';
```

Consulta correcta

```
SELECT *
FROM producto INNER JOIN fabricante
ON producto.codigo_fabricante = fabricante.codigo
WHERE fabricante.nombre = 'Lenovo';
```

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Errores comunes

3. Relacionamos las tablas utilizando nombres de columnas incorrectos.

Consulta incorrecta

```
SELECT *
FROM producto INNER JOIN fabricante
ON producto.codigo = fabricante.codigo;
```

Consulta correcta

```
SELECT *
FROM producto INNER JOIN fabricante
ON producto.codigo_fabricante = fabricante.codigo;
```

4. Cuando hacemos la intersección de tres tablas con INNER JOIN nos olvidamos de incluir ON en alguna de las intersecciones.

Consulta incorrecta

```
SELECT DISTINCT nombre_cliente, nombre, apellido1
FROM cliente INNER JOIN empleado
INNER JOIN pago
ON cliente.codigo_cliente = pago.codigo_cliente;
```

Consulta correcta

```
SELECT DISTINCT nombre_cliente, nombre, apellido1
FROM cliente INNER JOIN empleado
ON cliente.codigo_empleado_rep_ventas = empleado.codigo_empleado
INNER JOIN pago
ON cliente.codigo_cliente = pago.codigo_cliente;
```

CONSULTAS SOBRE VARIAS TABLAS. COMPOSICIÓN INTERNA Y CRUZADA

Consultas sobre varias tablas. Errores comunes

1. Cuando estamos usando LEFT JOIN o RIGHT JOIN no deberíamos tener varias condiciones en la cláusula ON.

Consulta incorrecta

```
SELECT *
FROM fabricante LEFT JOIN producto
ON fabricante.codigo = producto.codigo_fabricante AND  producto.codigo_fabricante IS NULL;
```

Consulta correcta.

```
SELECT *
FROM fabricante LEFT JOIN producto
ON fabricante.codigo = producto.codigo_fabricante
WHERE producto.codigo_fabricante IS NULL;
```

SUBCONSULTAS

Subconsultas

Una subconsulta es una consulta anidada dentro de otra consulta.

Debe tener en cuenta que no existe una única solución para resolver una consulta en SQL. En esta unidad vamos a estudiar cómo podemos resolver haciendo uso de subconsultas, algunas de las consultas que hemos resuelto en las unidades anteriores.

Tipos de subconsultas

El estándar SQL define tres tipos de subconsultas:

- Subconsultas de **fila**. Son aquellas que devuelven más de una columna pero una única fila.
- Subconsultas de **tabla**. Son aquellas que devuelve una o varias columnas y cero o varias filas.
- Subconsultas **escalares**. Son aquellas que devuelven una columna y una fila.

Subconsultas en la cláusula WHERE

Por ejemplo, suponga que queremos conocer el nombre del producto que tiene el mayor precio. En este caso podríamos realizar una primera consulta para buscar cuál es el valor del precio máximo y otra segunda consulta para buscar el nombre del producto cuyo precio coincide con el valor del precio máximo. La consulta sería la siguiente:

```
SELECT nombre
FROM producto
WHERE precio = (SELECT MAX(precio) FROM producto)
```

En este caso sólo hay un nivel de anidamiento entre consultas pero pueden existir varios niveles de anidamiento.

SUBCONSULTAS

Subconsultas

Subconsultas en la cláusula HAVING

Ejemplo: Devuelve un listado con todos los nombres de los fabricantes que tienen el mismo número de productos que el fabricante Asus.

```
SELECT fabricante.nombre, COUNT(producto.codigo)
FROM fabricante INNER JOIN producto
ON fabricante.codigo = producto.codigo_fabricante
GROUP BY fabricante.codigo
HAVING COUNT(producto.codigo) >= (
    SELECT COUNT(producto.codigo)
    FROM fabricante INNER JOIN producto
    ON fabricante.codigo = producto.codigo_fabricante
    WHERE fabricante.nombre = 'Asus');
```

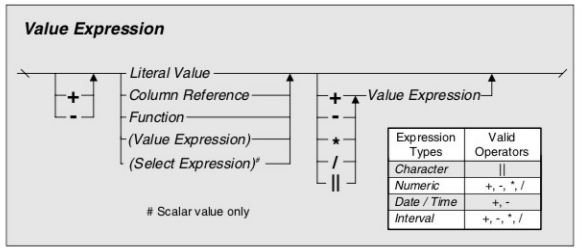
Subconsultas en la cláusula FROM

Ejemplo: Devuelve una listado de todos los productos que tienen un precio mayor o igual al precio medio de todos los productos de su mismo fabricante.

```
SELECT *
FROM producto INNER JOIN (
    SELECT codigo_fabricante, AVG(precio) AS media
    FROM producto
    GROUP BY codigo_fabricante) AS t
ON producto.codigo_fabricante = t.codigo_fabricante
WHERE producto.precio >= t.media;
```

Subconsultas en la cláusula SELECT

Las subconsultas que pueden aparecer en la cláusula SELECT tienen que ser subconsultas de tipo **escalar**, que devuelven una única fila y columna.



SUBCONSULTAS

Operadores que podemos usar en las subconsultas

Los operadores que podemos usar en las subconsultas son los siguientes:

- Operadores básicos de comparación (>,>=, <, <=, !=, <>, =).
- Predicados ALL y ANY.
- Predicado IN y NOT IN.
- Predicado EXISTS y NOT EXISTS.

Operadores básicos de comparación

Los operadores básicos de comparación (>,>=, <, <=, !=, <>, =) se pueden usar cuando queremos comparar una expresión con el valor que devuelve una subconsulta.

Los operadores básicos de comparación los vamos a utilizar para realizar comparaciones con subconsultas que devuelven un único valor, es decir, una columna y una fila.

SUBCONSULTAS

Operadores que podemos usar en las subconsultas

Operadores básicos de comparación

Ejemplo: Devuelve todos los productos de la base de datos que tienen un precio mayor o igual al producto más caro del fabricante Asus.

```
SELECT *
FROM producto
WHERE precio >= (SELECT MAX(precio)
    FROM fabricante INNER JOIN producto
    ON fabricante.codigo = producto.codigo_fabricante
    WHERE fabricante.nombre = 'Asus');
```

La consulta anterior también se puede escribir con subconsultas sin hacer uso de INNER JOIN.

```
SELECT *
FROM producto
WHERE precio = (
  SELECT MAX(precio)
  FROM producto
  WHERE codigo_fabricante = (
    SELECT codigo
    FROM fabricante
    WHERE nombre = 'Asus'));
```

SUBCONSULTAS

Operadores que podemos usar en las subconsultas

Subconsultas con ALL y ANY

ALL y ANY se utilizan con los operadores de comparación (>,>=, <, <=, !=, <>, =) y nos permiten comparar una expresión con el conjunto de valores que devuelve una subconsulta.

ALL y ANY los vamos a utilizar para realizar comparaciones con subconsultas que pueden devolver varios valores, es decir, una columna y varias filas.

Ejemplo: Podemos escribir la consulta que devuelve todos los productos de la base de datos que tienen un precio mayor o igual al producto más caro del fabricante Asus, haciendo uso de ALL. Por lo tanto estas dos consultas darían el mismo resultado.

```
SELECT *
FROM fabricante INNER JOIN producto
ON fabricante.codigo = producto.codigo_fabricante
WHERE precio >= (SELECT MAX(precio)
   FROM fabricante INNER JOIN producto
   ON fabricante.codigo = producto.codigo_fabricante
   WHERE fabricante.nombre = 'Asus');
```

```
SELECT *
FROM fabricante INNER JOIN producto
ON fabricante.codigo = producto.codigo_fabricante
WHERE precio >= ALL (SELECT precio
   FROM fabricante INNER JOIN producto
   ON fabricante.codigo = producto.codigo_fabricante
   WHERE fabricante.nombre = 'Asus');
```

La palabra reservada SOME es un alias de ANY. Por lo tanto, las siguientes consultas devolverían el mismo resultado:

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);

SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

SUBCONSULTAS

Operadores que podemos usar en las subconsultas

Subconsultas con IN y NOT IN

IN y NOT IN nos permiten comprobar si un valor está o no incluido en un conjunto de valores, que puede ser el conjunto de valores que devuelve una subconsulta.

IN y NOT IN los vamos a utilizar para realizar comparaciones con subconsultas que pueden devolver varios valores, es decir, una columna y varias filas.

Ejemplo: Devuelve un listado de los clientes que no han realizado ningún pedido.

```
SELECT *
FROM cliente
WHERE id NOT IN (SELECT id_cliente FROM pedido);
```

Cuando estamos trabajando con subconsultas, IN y = ANY realizan la misma función. Por lo tanto, las siguientes consultas devolverían el mismo resultado:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);

SELECT s1 FROM t1 WHERE s1 IN  (SELECT s1 FROM t2);
```

Ocurre lo mismo con NOT IN y <> ALL. Por lo tanto, las siguientes consultas devolverían el mismo resultado:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);

SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

SUBCONSULTAS

Operadores que podemos usar en las subconsultas

Subconsultas con IN y NOT IN

Importante:

Tenga en cuenta que cuando hay un valor NULL en el resultado de la consulta interna, la consulta externa no devuelve ningún valor.

Ejemplo: Devuelve un listado con el nombre de los departamentos que no tienen empleados asociados.

```
SELECT nombre
FROM departamento
WHERE codigo NOT IN (
  SELECT codigo_departamento
FROM empleado);
```

La consulta interna SELECT codigo_departamento FROM empleado, devuelve algunas filas con valores NULL y por lo tanto la consulta externa no devuelve ningún valor.

La forma de solucionarlo sería quitando los valores NULL de la consulta interna:

```
SELECT nombre
FROM departamento
WHERE codigo NOT IN (
  SELECT codigo_departamento
  FROM empleado
 WHERE codigo_departamento IS NOT NULL);
```

Subconsultas con EXISTS y NOT EXISTS

Ejemplo: Devuelve un listado de los clientes que no han realizado ningún pedido.

```
SELECT *
FROM cliente
WHERE NOT EXISTS (SELECT id_cliente FROM pedido WHERE cliente.id = pedido.id_cliente);
```