[Teoría] El lenguaje DML - en MySQL

Descripción

Tabla de contenidos

1. DML - El Lenguaje de Manipulación de Datos

2. La Sentencia INSERT.

3. La Sentencia UPDATE.

4. La Sentencia DELETE.

5. La Sentencia SELECT.
5.1. Cálculos / Cálculos aritméticos
5.2. Las Consultas Resumen [Agregación]
5.3. Subconsultas
5.4. Consultas multitabla. Composición Interna
5.5. Consultas multitabla. Composición Externa

1. DML - El Lenguaje de Manipulación de Datos

El **Lenguaje de Manipulación de Datos** (DML) es un componente fundamental en MySQL y en muchos otros sistemas de gestión de bases de datos relacionales. El DML se utiliza para manipular los datos almacenados en las tablas de una base de datos, permitiendo realizar operaciones como la inserción, selección, modificación y eliminación de registros. Es importante destacar que cualquier ejecución de un comando en un Sistema de Gestión de Bases de Datos (SGBD) se considera una consulta.

Además, Es importante tener en cuenta que en el contexto de los SGBD, el término derivado anglosajón más comúnmente utilizado para referirse a una consulta es "query". Sin embargo, es crucial entender que, aunque se use el término "consulta", cada ejecución de un comando en un SGBD, ya sea para insertar, seleccionar, modificar o eliminar datos, **se considera más como una "orden"** que una simple consulta pasiva. Estas órdenes tienen un impacto directo en la base de datos y pueden modificar su estado.

Sentencias del DML en MySQL:

1. INSERT:

La sentencia INSERT se utiliza para agregar nuevos registros a una tabla existente en la base de datos. Permite especificar los valores para cada columna en el nuevo registro que se va a insertar.

```
INSERT INTO tabla (columna1, columna2, ...) VALUES (valor1, valor2, ...);
```

2. SELECT:

La sentencia SELECT se utiliza para recuperar datos de una o varias tablas en la base de datos. Permite filtrar, ordenar y limitar los resultados según los criterios especificados.

```
SELECT columna1, columna2, ... FROM tabla WHERE condición;
```

3. DELETE:

La sentencia DELETE se utiliza para eliminar uno o varios registros de una tabla en la base de datos. Puede incluir una condición para especificar qué registros deben eliminarse.

```
DELETE FROM tabla WHERE condición;
```

4. UPDATE:

La sentencia UPDATE se utiliza para modificar los valores de uno o varios registros en una tabla existente en la base de datos. Permite actualizar los valores de las columnas según los criterios especificados.

```
UPDATE tabla SET columna1 = valor1, columna2 = valor2 WHERE condición;
```

2. La Sentencia INSERT.

En este capítulo, exploraremos en detalle la sentencia INSERT del Lenguaje de Manipulación de Datos (DML) en MySQL. La sentencia INSERT se utiliza para agregar nuevos registros a una tabla existente en la base de datos. Abordaremos la diferencia entre especificar y no especificar las columnas, el uso del DEFAULT en los campos y qué ocurre si se proporcionan más campos de los esperados en la lista de valores de la tabla.

1.0. Sintaxis:

Aquí está la sintaxis completa del comando INSERT:

```
INSERT INTO nombre_de_la_tabla (lista_de_columnas)
VALUES (valor1, valor2, ...);
```

1. **INSERT INTO**: Esta es la primera parte de la sintaxis y es obligatoria. Indica que estamos realizando una inserción de datos en una tabla específica.
2. **Nombre de la tabla**: Después de INSERT INTO, se debe especificar el nombre de la tabla en la que se insertarán los datos.
3. **Lista de columnas (opcional)**: A continuación del nombre de la tabla, opcionalmente se puede especificar una lista de columnas entre paréntesis. Esto indica en qué columnas se insertarán los valores. Si se omite esta parte, se insertarán valores en todas las columnas de la tabla en el orden en que se crearon.
4. **Valores a insertar**: Después de la lista de columnas (si se proporciona), se utiliza la palabra clave VALUES seguida de una lista de valores entre paréntesis. Estos valores deben coincidir en número y tipo con las columnas especificadas en la lista de columnas (si se proporciona) o con todas las columnas de la tabla en el orden en que se crearon.

1.1. Insertar registros especificando las columnas:

Cuando se insertan registros en una tabla especificando las columnas, se debe proporcionar un valor para cada columna mencionada en la lista de columnas.

```
INSERT INTO Personajes (nombre, raza, nivel_poder)
VALUES ('Goku', 'Saiyan', 9000);
```

Resultado esperado:

```
+----+-------+---------+-------------+
| id | nombre|  raza   | nivel_poder |
+----+-------+---------+-------------+
| 1  | Goku  | Saiyan  | 9000        |
+----+-------+---------+-------------+
```

1.2. Insertar registros sin especificar las columnas:

Cuando no se especifican las columnas, se deben proporcionar valores para todas las columnas de la tabla en el mismo orden en que fueron definidas.

```
INSERT INTO Personajes
VALUES (2, 'Vegeta', 'Saiyan', 8500);
```

Resultado esperado:

```
+----+--------+--------+-------------+
| id | nombre |  raza  | nivel_poder |
+----+--------+--------+-------------+
| 1  | Goku   | Saiyan | 9000        |
| 2  | Vegeta | Saiyan | 8500        |
+----+--------+--------+-------------+
```

1.3. Uso de DEFAULT en campos:

Algunas columnas pueden tener un valor predeterminado especificado en la definición de la tabla. En este caso, se puede usar la palabra clave DEFAULT para insertar el valor predeterminado.

```
INSERT INTO Personajes (nombre, raza)
VALUES ('Piccolo', DEFAULT);
```

Resultado esperado:

```
+----+--------+----------+-------------+
| id | nombre |   raza   | nivel_poder |
+----+--------+----------+-------------+
| 1  | Goku   | Saiyan   | 9000        |
| 2  | Vegeta | Saiyan   | 8500        |
| 3  | Piccolo| Namekian | NULL        |
+----+--------+----------+-------------+
```

1.4. Proporcionar más campos que la lista de valores de la tabla:

Si se proporcionan más campos de los esperados en la lista de valores, se generará un error.

```
INSERT INTO Personajes (nombre, raza, nivel_poder, planeta_origen)
VALUES ('Freezer', 'Alien', 12000, 'Planeta Namek');
```

Resultado esperado:

```
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

La sentencia INSERT es una herramienta poderosa para agregar datos a una tabla en MySQL. Al comprender cómo usarla correctamente y considerar las diferentes situaciones que pueden surgir, los usuarios pueden manipular eficazmente los datos en sus bases de datos.

1.5. Añadir varios registros con una sola instrucción INSERT:

En MySQL, es posible insertar múltiples registros en una tabla utilizando una sola instrucción INSERT. Esto puede ser útil cuando se necesita insertar varios registros de una vez en lugar de hacerlo uno por uno. Para lograr esto, se utiliza la siguiente sintaxis:

```
INSERT INTO nombre_de_la_tabla (columna1, columna2, ...)
VALUES
    (valor1, valor2, ...),
    (valor1, valor2, ...),
    (valor1, valor2, ...),
    ...;
```

Cada conjunto de valores entre paréntesis representa un registro distinto que se insertará en la tabla. Se pueden especificar los valores para todas las columnas o solo para un subconjunto de columnas, siempre y cuando los valores proporcionados coincidan en tipo y número con las columnas de la tabla.

Ejemplo:

Supongamos que mantenemos el uso en la siguiente tabla `Personajes`:

```
+----+--------+--------+-------------+
| id | nombre |  raza  | nivel_poder |
+----+--------+--------+-------------+
```

Y queremos insertar varios personajes nuevos en una sola instrucción:

```
INSERT INTO Personajes (nombre, raza, nivel_poder)
VALUES
    ('Gohan', 'Saiyan', 8000),
    ('Krillin', 'Humano', 2000),
    ('Vegeta', 'Saiyan', 8500);
```

Resultado esperado:

```
+----+--------+----------+-------------+
| id | nombre |   raza   | nivel_poder |
+----+--------+----------+-------------+
| 1  | Goku   | Saiyan   | 9000        |
| 2  | Vegeta | Saiyan   | 8500        |
| 3  | Piccolo| Namekian | NULL        |
| 4  | Gohan  | Saiyan   | 8000        |
| 5  | Krillin| Humano   | 2000        |
| 6  | Vegeta | Saiyan   | 8500        |
+----+--------+----------+-------------+
```

Observamos que se han insertado tres nuevos registros en la tabla `Personajes` con una sola instrucción INSERT. Cada registro se ha especificado con los valores correspondientes para las columnas `nombre`, `raza` y `nivel_poder`.

3. La Sentencia UPDATE.

En este capítulo, exploraremos en detalle la sentencia UPDATE en MySQL. La sentencia UPDATE se utiliza para modificar los valores de uno o varios registros en una tabla existente en la base de datos. Veremos su sintaxis, cómo realizar cambios en una sola columna o en múltiples columnas, y la importancia del filtro WHERE para especificar qué registros deben ser modificados.

La sentencia UPDATE es una instrucción del Lenguaje de Manipulación de Datos (DML) que **permite modificar los valores de uno o varios registros** en una tabla existente en una base de datos MySQL. Se utiliza para actualizar los datos existentes y mantener la integridad y precisión de la información almacenada en la base de datos.

1. ¿Qué es y para qué sirve la sentencia UPDATE?

La sentencia UPDATE es una instrucción del Lenguaje de Manipulación de Datos (DML) que permite modificar los valores de uno o varios registros en una tabla existente en una base de datos MySQL. Se utiliza para actualizar los datos existentes y mantener la integridad y precisión de la información almacenada en la base de datos.

2. Sintaxis de la sentencia UPDATE:

La sintaxis básica de la sentencia UPDATE es la siguiente:

```
UPDATE nombre_de_la_tabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condición;
```

- **nombre_de_la_tabla**: Especifica el nombre de la tabla en la que se realizarán las actualizaciones.
- **columna1, columna2, ...**: Son las columnas que se van a actualizar, junto con los nuevos valores que se les asignarán.
- **valor1, valor2, ...**: Son los nuevos valores que se asignarán a las columnas especificadas.
- **WHERE condición**: Es opcional y se utiliza para especificar qué registros deben ser actualizados. Si no se proporciona, la actualización se aplicará a todos los registros de la tabla.

3. Ejemplo de cambio en una columna:

Supongamos que queremos cambiar el nivel de poder de Goku a 9500:

```
UPDATE Personajes
SET nivel_poder = 9500
WHERE nombre = 'Goku';


Resultado esperado:
+----+--------+--------+-------------+
| id | nombre |  raza  | nivel_poder |
+----+--------+--------+-------------+
| 1  | Goku   | Saiyan | 9500        |
| 2  | Vegeta | Saiyan | 8500        |
| 3  | Gohan  | Saiyan | 8000        |
+----+--------+--------+-------------+
```

4. Ejemplo de cambio en múltiples columnas:

Supongamos que queremos cambiar **el nivel de poder** y **la raza** de Vegeta:

```
UPDATE Personajes
SET nivel_poder = 9000, raza = 'Saiyan de élite'
WHERE nombre = 'Vegeta';

Resultado esperado:
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 8000        |
+----+--------+--------------+-------------+
```

5. ¿Qué es el filtro WHERE y para qué se usa?

El filtro WHERE es una cláusula opcional que se utiliza para especificar qué registros deben ser actualizados. Permite realizar actualizaciones condicionales, aplicando los cambios solo a los registros que cumplan con la condición especificada.

En otras palabras el filtro **WHERE se utiliza para especificar qué filas deben ser actualizadas en una instrucción UPDATE**. Permite actualizar registros específicos basados en una condición.

Ejemplo con filtro WHERE:

Supongamos que queremos aumentar el nivel de poder de todos los Saiyan en 1000 unidades:

```
UPDATE Personajes
SET nivel_poder = nivel_poder + 1000
WHERE raza = 'Saiyan';

Resultado esperado:
```

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 10000       |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 10000       |
+----+--------+--------------+-------------+
```

NO USO del filtro WHERE:

Si no se proporciona un filtro WHERE, la actualización se aplicará a todos los registros de la tabla:

```
UPDATE Personajes
SET nivel_poder = 11000;

Resultado esperado:
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 11000       |
| 2  | Vegeta | Saiyan Elite | 11000       |
| 3  | Gohan  | Saiyan       | 11000       |
+----+--------+--------------+-------------+
```

Observamos que todos los personajes tienen ahora el mismo nivel de poder debido a la falta de una condición WHERE en la sentencia UPDATE.

4. La Sentencia DELETE.

En este capítulo, exploraremos la sentencia DELETE en MySQL, que se utiliza para eliminar filas de una tabla. Veremos qué es y para qué sirve, su sintaxis, cómo eliminar registros utilizando el filtro WHERE, y el efecto de no utilizar un filtro WHERE en una instrucción DELETE.

1. ¿Qué es la sentencia DELETE y para qué sirve?

La sentencia DELETE se utiliza para eliminar una o varias filas de una tabla en una base de datos. Sirve para eliminar registros que ya no son necesarios o que son incorrectos.

2. Sintaxis de la sentencia DELETE:

La sintaxis básica de la sentencia DELETE es la siguiente:

```
DELETE FROM nombre_de_la_tabla WHERE condición;
```

- `nombre_de_la_tabla`: Especifica el nombre de la tabla de la que se eliminarán los registros.
- `WHERE condición`: Es opcional y se utiliza para especificar qué filas se eliminarán. Si se omite, se eliminarán todas las filas de la tabla.

3. Ejemplo: Eliminar registros utilizando el filtro WHERE

Supongamos que queremos eliminar a todos los personajes que tengan un nivel de poder menor a 8000 en la siguiente tabla:

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 8000        |
+----+--------+--------------+-------------+

mysql> DELETE FROM Personajes WHERE nivel_poder < 8000;
```

Resultado esperado:

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
+----+--------+--------------+-------------+
```

4. Ejemplo: No utilizar un filtro WHERE

Supongamos que olvidamos agregar el filtro WHERE y queremos eliminar todos los personajes de la siguiente tabla:

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 8000        |
+----+--------+--------------+-------------+
```

```
mysql> DELETE FROM Personajes;
```

Resultado esperado:

```
+----+--------+------+-------------+
| id | nombre | raza | nivel_poder |
+----+--------+------+-------------+
+----+--------+------+-------------+
```

Observamos que todos los personajes han sido eliminados de la tabla debido a la falta de una condición WHERE en la sentencia DELETE. Es importante tener cuidado al omitir el filtro WHERE para evitar eliminar accidentalmente todos los registros de una tabla.

5. La Sentencia SELECT.

En este capítulo, exploraremos los fundamentos de la sentencia SELECT en MySQL. Cubriremos su importancia en SQL, su sintaxis básica y cómo seleccionar columnas específicas de una tabla.

1. Introducción a la sentencia SELECT

La sentencia SELECT es una de las más fundamentales en SQL y se utiliza para recuperar datos de una o más tablas de una base de datos. Es una herramienta poderosa que nos permite consultar y analizar la información almacenada en una base de datos.

2. Sintaxis básica de la sentencia SELECT

La sintaxis básica de la sentencia SELECT es la siguiente:

```
SELECT column1, column2, ...
FROM table_name;
```

- `column1, column2, ...`: Especifica las columnas que deseamos seleccionar de la tabla.
- `table_name`: Especifica la tabla de la que queremos seleccionar los datos.

Selección de todas las columnas de la tabla Personajes

Supongamos que tenemos una tabla llamada `Personajes` con las siguientes columnas: `id`, `nombre`, `raza` y `nivel_poder`. Para seleccionar todas las columnas de esta tabla, usaríamos la siguiente sentencia SELECT:

```
SELECT *
FROM Personajes;
```

Resultado esperado:

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 8000        |
+----+--------+--------------+-------------+
```

Selección de columnas específicas.

También podemos seleccionar columnas específicas en lugar de todas las columnas utilizando la sintaxis de la sentencia SELECT. Por ejemplo:

```
SELECT nombre, raza
FROM Personajes;
```

Resultado esperado:

```
+--------+--------------+
| nombre |     raza     |
+--------+--------------+
| Goku   | Saiyan       |
| Vegeta | Saiyan Elite |
| Gohan  | Saiyan       |
+--------+--------------+
```

En este ejemplo, solo se seleccionan las columnas `nombre` y `raza` de la tabla `Personajes`.

En este capítulo hemos introducido los conceptos básicos de la sentencia SELECT en MySQL, incluyendo su importancia, su sintaxis básica y cómo seleccionar columnas específicas de una tabla. En los próximos capítulos, profundizaremos en aspectos más avanzados de la sentencia SELECT y exploraremos cómo filtrar, ordenar y agrupar los resultados de una consulta.

Concatenación de columnas.

Se parte de la siguiente tabla en la que se indica si se ha alcanzado el estado SuperSaiyan con la columna super:

```
+--------+--------------+-------+
| nombre |    raza      | super |
+--------+--------------+-------+
| Goku   | Saiyan       | Super |
| Vegeta | Saiyan Elite |       |
| Gohan  | Saiyan       | Super |
+--------+--------------+-------+
```

Se puede concatenar el resultado de una consulta y mostrarlo en la propia consulta SELECT (sin afectar a la tabla origen) para mostrar las columnas `nombre` y una concatenación de las columnas `super` y `raza`:

```
SELECT nombre, CONCAT(super, '-', raza) AS super_raza_concatenado
FROM Personajes;
```

Resultado esperado:

```
+--------+----------------------+
| nombre | super_raza_concatenado|
+--------+----------------------+
| Goku   | Super-Saiyan         |
| Vegeta | -Saiyan Elite        |
| Gohan  | Super-Saiyan         |
+--------+----------------------+
```

En este resultado, la columna `nombre` se muestra independientemente, mientras que la concatenación de las columnas `super` y `raza` se muestra en una sola columna llamada `super_raza_concatenado`.

o también si no se necesita el guión separador:

```
SELECT nombre, CONCAT(super, raza) AS super_raza_concatenado
FROM Personajes;
```

Resultado esperado:

```
+--------+----------------------+
| nombre | super_raza_concatenado|
+--------+----------------------+
| Goku   | SuperSaiyan          |
| Vegeta | Saiyan Elite         |
| Gohan  | SuperSaiyan          |
+--------+----------------------+
```

En este resultado, la columna `nombre` se muestra independientemente, mientras que la concatenación de las columnas `super` y `raza` se muestra en una sola columna llamada `super_raza_concatenado`.

Operaciones matemáticas.

En las consultas se pueden realizar operaciones matemáticas:

```
SELECT 1 + 5;
```

Resultado esperado:

```
+------+
| 1+5  |
+------+
| 6    |
+------+
```

o también

```
SELECT 1 + 5 AS suma;
```

Resultado esperado:

```
+------+
| suma |
+------+
| 6    |
+------+
```

En este ejemplo, la consulta SELECT realiza la operación matemática 1 + 5 y devuelve el resultado en una columna llamada `suma`. El resultado de la suma es 6.

Valores únicos DISTINCT.

Supongamos que tenemos una tabla llamada `Personajes` con una columna llamada `raza` con la siguiente estructura y datos:

```
+--------+--------------+
| nombre |     raza     |
+--------+--------------+
| Goku   | Saiyan       |
| Vegeta | Saiyan Elite |
| Gohan  | Saiyan       |
| Piccolo| Namekiano    |
| Goku   | Saiyan       |
+--------+--------------+
```

Queremos obtener una lista de todas las razas únicas de los personajes.
Consulta SQL:

```
SELECT DISTINCT raza
FROM Personajes;
```

Resultado esperado:

```
+--------------+
|     raza     |
+--------------+
| Saiyan       |
| Saiyan Elite |
| Namekiano    |
+--------------+
```

En este ejemplo, la consulta SELECT utiliza la cláusula DISTINCT para seleccionar solo los valores únicos de la columna `raza` de la tabla `Personajes`. Como resultado, obtenemos una lista de todas las razas únicas de los personajes sin duplicados.

3. Sintaxis COMPLETA de la sentencia SELECT

La sintaxis general de la sentencia SELECT es la siguiente:

```
SELECT column1, column2, ...
FROM table_name
[WHERE condition]
[GROUP BY column1, column2, ...]
[HAVING condition]
[ORDER BY column1, column2, ... [ASC | DESC]];
```

Desglose de las cláusulas:

1. **SELECT:**
   - La cláusula SELECT se utiliza para especificar las columnas que se desean seleccionar en la consulta.
   - Puede contener una lista de nombres de columnas separados por comas, o el asterisco (*) para seleccionar todas las columnas.
   - Ejemplo: `SELECT nombre, raza FROM Personajes;`
2. **FROM:**
   - La cláusula FROM se utiliza para especificar la tabla o tablas de donde se van a seleccionar los datos.
   - Puede contener el nombre de una tabla o una lista de tablas separadas por comas si se desean combinar datos de múltiples tablas.
   - Ejemplo: `SELECT * FROM Personajes;`
3. **WHERE:**
   - La cláusula WHERE se utiliza para filtrar filas basadas en una condición específica.
   - Es opcional y se coloca después de la cláusula FROM.
   - Puede contener condiciones utilizando operadores de comparación como =, <>, <, >, <=, >=, así como operadores lógicos como AND, OR y NOT.
   - Ejemplo: `SELECT * FROM Personajes WHERE nivel_poder > 9000;`
4. **GROUP BY:**
   - La cláusula GROUP BY se utiliza para agrupar filas que tienen el mismo valor en una o más columnas.
   - Se utiliza junto con funciones de agregación como SUM, AVG, COUNT, MIN y MAX para realizar cálculos en grupos de filas.
   - Ejemplo: `SELECT raza, COUNT(*) FROM Personajes GROUP BY raza;`
5. **HAVING:**
   - La cláusula HAVING se utiliza para filtrar grupos de filas devueltos por la cláusula GROUP BY.
   - Funciona de manera similar a la cláusula WHERE, pero se aplica después de la agrupación de filas.
   - Ejemplo: `SELECT raza, COUNT(*) FROM Personajes GROUP BY raza HAVING COUNT(*) > 1;`
6. **ORDER BY:**

- La cláusula ORDER BY se utiliza para ordenar los resultados de la consulta basados en una o más columnas.
- Puede ordenar los resultados de manera ascendente (ASC) o descendente (DESC).
- Ejemplo: `SELECT * FROM Personajes ORDER BY nivel_poder DESC;`

Esta es la sintaxis básica de la sentencia SELECT en MySQL, que puede ser combinada y modificada para realizar consultas más complejas y específicas según sea necesario.

Ejemplo de SELECT con WHERE:

Consulta SQL:

```
SELECT *
FROM Personajes
WHERE nivel_poder > 9000;
```

Tabla Personajes (antes de la consulta):

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 8000        |
+----+--------+--------------+-------------+
```

Resultado esperado:

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
+----+--------+--------------+-------------+
```

En este ejemplo, la cláusula WHERE filtra las filas de la tabla `Personajes` y selecciona solo aquellas donde el valor de la columna `nivel_poder` es mayor que 9000. Como resultado, obtenemos solo el personaje "Goku" con un nivel de poder de 9500.

Ejemplo con GROUP BY:

Consulta SQL:

```
SELECT raza, COUNT(*) AS cantidad_personajes
FROM Personajes
GROUP BY raza;
```

Tabla Personajes (antes de la consulta):

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 8000        |
| 4  | Piccolo| Namekiano    | 8500        |
+----+--------+--------------+-------------+
```

Resultado esperado:

```
+--------------+--------------------+
|     raza     | cantidad_personajes|
+--------------+--------------------+
| Saiyan       |         3          |
| Saiyan Elite |         1          |
| Namekiano    |         1          |
+--------------+--------------------+
```

En este ejemplo, la cláusula GROUP BY agrupa las filas de la tabla `Personajes` según el valor de la columna `raza`. Luego, la función de agregación COUNT(*) cuenta el número de filas en cada grupo, dando como resultado el número de personajes por cada raza.

Ejemplo con HAVING:

Consulta SQL:

```
SELECT raza, COUNT(*) AS cantidad_personajes
FROM Personajes
GROUP BY raza
HAVING COUNT(*) > 1;
```

Tabla Personajes (antes de la consulta):

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 8000        |
| 4  | Piccolo| Namekiano    | 8500        |
+----+--------+--------------+-------------+
```

Resultado esperado:

```
+--------------+--------------------+
|     raza     | cantidad_personajes|
+--------------+--------------------+
| Saiyan       |         3          |
+--------------+--------------------+
```

En este ejemplo, la cláusula HAVING se utiliza para filtrar los grupos de filas devueltos por la cláusula GROUP BY. La condición HAVING COUNT(*) > 1 indica que solo queremos mostrar las razas que tienen más de un personaje. Como resultado, obtenemos solo la raza "Saiyan", ya que es la única que cumple con esa condición.

Ejemplo con ORDER BY:

Consulta SQL:

```
SELECT *
FROM Personajes
ORDER BY nivel_poder DESC;
```

Tabla Personajes (antes de la consulta):

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 3  | Gohan  | Saiyan       | 8000        |
| 4  | Piccolo| Namekiano    | 8500        |
+----+--------+--------------+-------------+
```

Resultado esperado:

```
+----+--------+--------------+-------------+
| id | nombre |     raza     | nivel_poder |
+----+--------+--------------+-------------+
| 1  | Goku   | Saiyan       | 9500        |
| 2  | Vegeta | Saiyan Elite | 9000        |
| 4  | Piccolo| Namekiano    | 8500        |
| 3  | Gohan  | Saiyan       | 8000        |
+----+--------+--------------+-------------+
```

En este ejemplo, la cláusula ORDER BY se utiliza para ordenar los resultados de la consulta según la columna `nivel_poder`, en orden descendente (DESC). Como resultado, obtenemos los personajes ordenados por su nivel de poder de mayor a menor.

5.1. Cálculos / Cálculos aritméticos

En MySQL, la cláusula SELECT permite realizar cálculos aritméticos y manipular los valores de las columnas durante la recuperación de datos. Esto proporciona flexibilidad adicional para obtener resultados personalizados según los requisitos de la consulta. Aquí se detallan los aspectos clave relacionados con los cálculos en una consulta SELECT:

Operadores Aritméticos

Puedes realizar operaciones aritméticas en los valores de las columnas utilizando los operadores aritméticos estándar, como suma (+), resta (-), multiplicación (*) y división (/), dentro de la cláusula SELECT.

Ejemplo, se parte de la siguiente tabla:

```
+----+--------+----------+-------------+
| id | nombre |  raza    | nivel_poder |
+----+--------+----------+-------------+
| 1  | Goku   |  Saiyan  | 9000        |
| 2  | Vegeta |  Saiyan  | 8500        |
| 3  | Piccolo| Namekiano| 5000        |
+----+--------+----------+-------------+
```

```
SELECT nivel_poder * 100 AS nivel_poder_actualizado
FROM Guerreros;
```

Resultado:

```
+------------------------+
| nivel_poder_actualizado|
+------------------------+
| 900000                 |
| 850000                 |
| 500000                 |
+------------------------+
```

En este ejemplo, la consulta realiza un cálculo aritmético multiplicando el valor de la columna `nivel_poder` por 100 y devuelve el resultado en una nueva columna llamada `nivel_poder_actualizado`.

Si bien este ejemplo puede ser válido, vamos a ir ahora un paso más lejos para ver la verdadera potencia de los cálculos. Imagina ahora que se quiere mostrar una columna nueva con el valor del nivel de poder cuando el guerrero pasa a SuperSaiyan (multiplicando su nivel de poder base por 50). Observa el siguiente ejemplo:

```
SELECT nombre, nivel_poder, nivel_poder * 50 AS nivel_poder_supersaiyan
FROM Guerreros;
```

Resultado:

```
+--------+-------------+--------------------------+
| nombre | nivel_poder | nivel_poder_supersaiyan  |
+--------+-------------+--------------------------+
| Goku   | 9000        | 450000                   |
| Vegeta | 8500        | 425000                   |
| Piccolo| 5000        | 250000                   |
+--------+-------------+--------------------------+
```

En este ejemplo, la consulta selecciona las columnas `nombre` y `nivel_poder` de la tabla `Guerreros`, y realiza un cálculo aritmético para multiplicar el `nivel_poder` por 50, devolviendo el resultado en una nueva columna llamada `nivel_poder_supersaiyan`.

No obstante, en este ejemplo anterior falla algo, y es que Piccolo no tiene el estado SuperSaiyan porque es un namekiano. Por lo tanto al no tener nivel SuperSaiyan en la columna correspondiente debería mantener intacto su nivel de poder. Para ello se utiliza la estructura **CASE - WHEN - THEN - ELSE.** Observa el siguiente ejemplo que resuelve este problema:

```
SELECT nombre, nivel_poder,
       CASE
           WHEN raza = 'Saiyan' THEN nivel_poder * 50
           ELSE 'No SuperSaiyan'
       END AS nivel_poder_supersaiyan
FROM Guerreros;
```

El resultado mostraría algo como esto:

```
+---------+-------------+-------------------------+
| nombre  | nivel_poder | nivel_poder_supersaiyan |
+---------+-------------+-------------------------+
| Goku    |        9000 | 450000                  |
| Vegeta  |        8500 | 425000                  |
| Piccolo |        5000 | NO SuperSaiyan          |
+---------+-------------+-------------------------+
```

En este ejemplo, la consulta selecciona las columnas `nombre` y `nivel_poder` de la tabla `Guerreros`, y utiliza una expresión CASE para multiplicar el `nivel_poder` por 50 solo si el guerrero es de raza Saiyan. Si no, simplemente devuelve la cadena '`NO SuperSaiyan`'.

Operadores Relacionales.

Los operadores relacionales en SQL se utilizan para comparar dos valores y determinar la relación entre ellos. Aquí tienes algunos ejemplos utilizando la tabla `Guerreros`:

- **Operador `=` (igual)**: Se utiliza para verificar si dos valores son iguales.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder = 9000;
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros cuyo nivel de poder es exactamente 9000.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Goku    |        9000 |
+---------+-------------+
```

- **Operador `!=` o `<>` (distinto o diferente)**: Se utiliza para verificar si dos valores no son iguales.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder != 9000;
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros cuyo nivel de poder no es 9000.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Vegeta  |        8500 |
| Piccolo |        5000 |
+---------+-------------+
```

- **Operador `>` (mayor que)**: Se utiliza para verificar si un valor es mayor que otro.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder > 8000;
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros cuyo nivel de poder es mayor que 8000.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Goku    |        9000 |
| Vegeta  |        8500 |
+---------+-------------+
```

- **Operador `<` (menor que)**: Se utiliza para verificar si un valor es menor que otro.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder < 8000;
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros cuyo nivel de poder es menor que 8000.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Piccolo |        5000 |
+---------+-------------+
```

- **Operador `>=` (mayor o igual que)**: Se utiliza para verificar si un valor es mayor o igual que otro.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder >= 8500;
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros cuyo nivel de poder es mayor o igual que 8500.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Goku    |        9000 |
| Vegeta  |        8500 |
+---------+-------------+
```

- **Operador `<=` (menor o igual que)**: Se utiliza para verificar si un valor es menor o igual que otro.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder <= 8500;
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros cuyo nivel de poder es menor o igual que 8000.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Vegeta  |        8500 |
| Piccolo |        5000 |
+---------+-------------+
```

Resumiendo, a continuación tienes los operadores y su significado:

```
+------------+-----------------------------------+
| Operador   | Significado                       |
+------------+-----------------------------------+
| =          | Igual                             |
| != o <>    | Distinto o diferente              |
| >          | Mayor que                         |
| <          | Menor que                         |
| >=         | Mayor o igual que                 |
| <=         | Menor o igual que                 |
+------------+-----------------------------------+
```

Operadores Lógicos.

Los operadores lógicos en SQL se utilizan para combinar condiciones en las consultas y realizar operaciones lógicas sobre ellas. A continuación te dejo una explicación de los operadores lógicos con ejemplos (se mantiene el uso de la tabla `Guerreros`):

- **Operador `AND`**: Se utiliza para combinar dos o más condiciones y devuelve true si todas las condiciones son verdaderas.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE raza = 'Saiyan' AND nivel_poder > 8500;
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros que son de raza Saiyan y tienen un nivel de poder mayor que 8500.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Goku    |        9000 |
+---------+-------------+
```

- **Operador `OR`**: Se utiliza para combinar dos o más condiciones y devuelve true si al menos una de las condiciones es verdadera.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE raza = 'Saiyan' OR nivel_poder > 4000;
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros que son de raza Saiyan o tienen un nivel de poder mayor que 4000.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Goku    |        9000 |
| Vegeta  |        8500 |
| Piccolo |        5000 |
+---------+-------------+
```

- **Operador `NOT`**: Se utiliza para negar una condición y devuelve true si la condición es falsa.

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE NOT raza = 'Saiyan';
```

Este ejemplo selecciona el nombre y el nivel de poder de los guerreros que no son de raza Saiyan.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Piccolo |        5000 |
+---------+-------------+
```

Resumiendo, aquí tienes la tabla resumen de los operadores lógicos con su significado:

```
+-----------+-----------------------------------+
| Operador  | Significado                       |
+-----------+-----------------------------------+
| AND       | Devuelve true si todas las        |
|           | condiciones son verdaderas.       |
| OR        | Devuelve true si al menos una     |
|           | de las condiciones es verdadera.  |
| NOT       | Devuelve true si la condición     |
|           | es falsa.                         |
+-----------+-----------------------------------+
```

Cláusula BETWEEN

La sentencia `BETWEEN` en MySQL se utiliza para verificar si un valor se encuentra dentro de un rango especificado de valores. Por ejemplo, supongamos que tenemos una tabla llamada `poderes` que almacena información sobre los niveles de poder de varios guerreros de Dragon Ball Z. La tabla tiene las siguientes columnas: `nombre`, `nivel_poder`, `raza`, etc.

Tabla `poderes`:

```
+---------+-------------+-------------------+
| nombre  | nivel_poder |       raza        |
+---------+-------------+-------------------+
| Goku    |        9000 |           Saiyan  |
| Vegeta  |        8500 |           Saiyan  |
| Piccolo |        5000 |         Namekiano |
| Freezer |       12000 | Emperador del Mal |
| Cell    |        8500 |      Bio-Androide |
+---------+-------------+-------------------+
```

Por ejemplo podemos utilizar la sentencia `BETWEEN` para seleccionar los guerreros cuyo nivel de poder está dentro de un rango específico. Por ejemplo, podríamos querer seleccionar los guerreros cuyo nivel de poder está entre 8500 y 10000:

```

SELECT *
FROM poderes
WHERE nivel_poder BETWEEN 8500 AND 10000;
```

Esto devolvería los siguientes resultados:

```
+---------+-------------+--------------+
| nombre  | nivel_poder |     raza     |
+---------+-------------+--------------+
| Goku    |        9000 |      Saiyan  |
| Vegeta  |        8500 |      Saiyan  |
| Cell    |        8500 | Bio-Androide |
+---------+-------------+--------------+
```

En este ejemplo, Goku, Vegeta y Cell tienen un nivel de poder que está dentro del rango especificado (8500 a 10000).

Es importante tener en cuenta que la sentencia `BETWEEN` **es inclusiva**, lo que significa que incluye los valores límite (`8500` y `10000`). Si deseamos excluir los valores límite, debemos utilizar los operadores de comparación (`>` y `<`) en su lugar.

Fíjate que la consulta anterior usando el BETWEEN es equivalente a hacer la siguiente consulta:

```
-- SELECT *
-- FROM poderes
-- WHERE nivel_poder >= 8500 AND nivel_poder <= 10000;
```

Cláusula IN.

La sentencia `IN` en MySQL se utiliza para especificar múltiples valores en una condición WHERE. Permite verificar si un valor se encuentra dentro de un conjunto de valores especificados. Por otro lado, la sentencia `NOT IN` hace lo contrario: verifica si un valor no se encuentra dentro del conjunto de valores especificados.

Supongamos que tenemos una tabla llamada `guerreros` que almacena información sobre varios personajes de Dragon Ball Z, incluyendo su raza. Queremos seleccionar los guerreros que son Saiyan y Namekiano . Podemos usar la sentencia `IN` de la siguiente manera:

```
SELECT *
FROM guerreros
WHERE raza IN ('Saiyan', 'Namekiano');
```

Esto devolverá los siguientes resultados:

```
+---------+-------------+---------+
|  nombre |     raza    | poder   |
+---------+-------------+---------+
| Goku    |       Saiyan | 9000    |
| Vegeta  |       Saiyan | 8500    |
| Piccolo |     Namekiano| 5000    |
+---------+-------------+---------+
```

En este ejemplo, la sentencia `IN` nos permite seleccionar los guerreros cuya raza está incluida en el conjunto especificado ('Saiyan', 'Namekiano', 'Bio-Androide').

Por otro lado, si queremos seleccionar los guerreros que no son ni Saiyan ni Namekiano, podemos usar la sentencia `NOT IN`:

```
SELECT *
FROM guerreros
WHERE raza NOT IN ('Saiyan', 'Namekiano');
```

Esto devolverá los siguientes resultados:

```
+---------+-------------+---------+
|  nombre |     raza    | poder   |
+---------+-------------+---------+
| Cell    | Bio-Androide | 8500    |
+---------+-------------+---------+
```

En este ejemplo, la sentencia `NOT IN` nos permite seleccionar los guerreros cuya raza no está incluida en el conjunto especificado ('Saiyan', 'Namekiano').

Cláusula LIKE.

La sentencia `LIKE` en MySQL se utiliza para realizar búsquedas de patrones en una columna de texto. Funciona comparando cada valor de la columna con un patrón especificado y seleccionando las filas que coinciden con ese patrón. Aquí te explico cómo funciona esta sentencia sobre la tabla `guerreros`:

Supongamos que queremos buscar todos los guerreros cuyos nombres comienzan con "G". Podemos usar la sentencia `LIKE` de la siguiente manera:

```
SELECT *
FROM guerreros
WHERE nombre LIKE 'G%';
```

En esta consulta, `'G%'` es el patrón que estamos buscando. El símbolo **`%` es un comodín que representa cero o más caracteres**. Por lo tanto, la condición `LIKE 'G%'` seleccionará todas las filas donde el valor de la columna `nombre` comience con la letra "G".

El resultado de esta consulta sería:

```
+---------+---------+---------+
|  nombre |  raza   | poder   |
+---------+---------+---------+
| Goku    | Saiyan  | 9000    |
+---------+---------+---------+
```

Esto se debe a que solo el nombre "Goku" comienza con la letra "G" en la tabla `guerreros`.

También podemos buscar nombres que contengan una secuencia específica de caracteres en cualquier parte del nombre. Por ejemplo, si queremos buscar todos los guerreros cuyos nombres contienen "co", podemos hacerlo así:

```
SELECT *
FROM guerreros
WHERE nombre LIKE '%co%';
```

En este caso, `%co%` es el patrón que estamos buscando. El símbolo `%` antes y después de "co" indica que puede haber cero o más caracteres antes y después de "co". Por lo tanto, la condición `LIKE '%co%'` seleccionará todas las filas donde el valor de la columna `nombre` contenga la secuencia "co".

El resultado de esta consulta sería:

```
+---------+----------+---------+
|  nombre |   raza   | poder   |
+---------+----------+---------+
| Piccolo | Namekiano| 5000    |
+---------+----------+---------+
```

Esto se debe a que solo el nombre "Piccolo" contiene la secuencia "co" en la tabla `guerreros`.

En resumen, la sentencia `LIKE` en MySQL es una herramienta poderosa para buscar patrones en una columna de texto y seleccionar las filas que coinciden con esos patrones. Los comodines `%` permiten buscar patrones más flexibles y completos.

Sin embargo, no solo existe el carácter comodín %, también existe **el carácter comodín '_'** en la sentencia `LIKE` en MySQL. Éste **se utiliza para representar un solo carácter en un patrón de búsqueda**. Mientras que el comodín `%` representa cero o más caracteres, el comodín `_` representa exactamente un carácter en una posición específica del patrón.

Aquí te explico cómo funciona el carácter comodín `_` en la sentencia `LIKE` sobre la tabla `guerreros`:

Supongamos que queremos buscar un guerrero cuyo nombre tiene cinco letras y la tercera letra es una "k". Podemos usar el carácter comodín `_` de la siguiente manera:

```
SELECT *
FROM guerreros
WHERE nombre LIKE '__k%';
```

En esta consulta, `__k%` es el patrón que estamos buscando. El primer `_` representa cualquier carácter en la primera posición del nombre, el segundo `_` representa cualquier carácter en la segunda posición del nombre, y la `k` representa la letra "k" en la tercera posición del nombre. El `%` al final del patrón indica que puede haber cero o más caracteres después de la "k".

El resultado de esta consulta sería:

```
+---------+---------+---------+
|  nombre |  raza   | poder   |
+---------+---------+---------+
| Goku    | Saiyan  | 9000    |
+---------+---------+---------+
```

Esto se debe a que solo el nombre "**Go**ku" tiene cinco letras y la tercera letra es una "k" en la tabla `guerreros`.

En resumen, el carácter comodín `_` en la sentencia `LIKE` permite representar **un solo carácter en una posición específica del patrón de búsqueda**, lo que proporciona una mayor flexibilidad en las consultas de búsqueda de patrones.

Para buscar una serie de carácteres de los cuales pueden ser uno u otro, se deben de integrar dentro de dos corchetes, por ejemplo:

```
SELECT *
FROM Guerreros
WHERE nombre LIKE '%[aeiou]%';
```

En este ejemplo se está buscando cualquier nombre que en alguna parte de su cadena contenga una vocal (cualquier carácter de los que hay dentro de los corchetes).

Cláusula IS NULL & IS NOT NULL

La expresión `IS NULL` en MySQL se utiliza para verificar si un valor en una columna es nulo. Un valor nulo representa la ausencia de un valor conocido o la falta de información en una columna. Supongamos que queremos buscar guerreros cuyo nivel de poder no está definido, es decir, que la columna `nivel_poder` tiene un valor nulo. Podemos usar la expresión `IS NULL` de la siguiente manera:

```
SELECT *
FROM guerreros
WHERE nivel_poder IS NULL;
```

En esta consulta, `nivel_poder IS NULL` verifica si el valor en la columna `nivel_poder` es nulo. La expresión `IS NULL` devuelve verdadero si el valor es nulo y falso si no lo es. En este caso, la consulta devolverá todas las filas donde el nivel de poder no esté definido.

El resultado de esta consulta dependerá de los datos de la tabla `guerreros`, pero como en nuestra tabla de ejemplo todos los personajes tienen un nivel de poder definido, el resultado sería algo similar a esto:

```
+---------+------------+------------+
|  nombre |   raza     | nivel_poder|
+---------+------------+------------+
|         |            |            |
+---------+------------+------------+
```

Una tabla vacía.

El caso contrario sería utilizar la expresión `IS NOT NULL` para encontrar aquellos guerreros cuyo nivel de poder está definido, es decir, aquellos que no tienen un valor nulo en la columna `nivel_poder`.

```
SELECT *
FROM guerreros
WHERE nivel_poder IS NOT NULL;
```

En esta consulta, `nivel_poder IS NOT NULL` verifica si el valor en la columna `nivel_poder` no es nulo. La expresión `IS NOT NULL` devuelve verdadero si el valor no es nulo y falso si lo es. En este caso, la consulta devolverá todas las filas porque todos los guerreros tienen su poder definido.

```
+---------+-------------+-------------------+
| nombre  | nivel_poder |       raza        |
+---------+-------------+-------------------+
| Goku    |        9000 |           Saiyan  |
| Vegeta  |        8500 |           Saiyan  |
| Piccolo |        5000 |         Namekiano |
| Freezer |       12000 | Emperador del Mal |
| Cell    |        8500 |      Bio-Androide |
+---------+-------------+-------------------+
```

5.2. Las Consultas Resumen [Agregación]

Las consultas de resumen o de agregación en MySQL son aquellas que nos permiten calcular y obtener información resumida sobre un conjunto de datos. Estas consultas son útiles para obtener estadísticas o información agregada de una tabla. Algunas de las funciones de resumen más comunes en MySQL son:

1. **COUNT()**: Esta función cuenta el número de filas que cumplen cierta condición. Puede contar todas las filas (`COUNT(*)`) o contar las filas en las que una columna específica no es nula (`COUNT(columna)`).
2. **SUM()**: Calcula la suma de los valores en una columna numérica.
3. **AVG()**: Calcula el promedio de los valores en una columna numérica.
4. **MIN()**: Encuentra el valor mínimo en una columna.
5. **MAX()**: Encuentra el valor máximo en una columna.

La sintaxis básica para utilizar estas funciones es la siguiente:

```
SELECT COUNT(*), COUNT(columna), SUM(columna), AVG(columna), MIN(columna), MAX(columna)
FROM tabla
WHERE condicion;
```

Aquí hay una explicación de cada función:

- `COUNT(*)`: Cuenta todas las filas de la tabla que cumplen la condición especificada.
- `COUNT(columna)`: Cuenta las filas en las que el valor de la columna especificada no es nulo.
- `SUM(columna)`: Calcula la suma de los valores en la columna especificada.
- `AVG(columna)`: Calcula el promedio de los valores en la columna especificada.
- `MIN(columna)`: Encuentra el valor mínimo en la columna especificada.
- `MAX(columna)`: Encuentra el valor máximo en la columna especificada.

Estas funciones pueden combinarse con la cláusula `GROUP BY` para calcular resúmenes para grupos específicos de filas, y también pueden utilizarse con la cláusula `HAVING` para filtrar los resultados de la función de resumen.

Para los siguientes ejemplos, Imagina la siguiente tabla Guerreros2:

```
+----+---------+--------------------+-------------+
| id | nombre  | raza               | nivel_poder |
+----+---------+--------------------+-------------+
| 1  | Goku    | Saiyan             | 9000        |
| 2  | Vegeta  | Saiyan             | 8500        |
| 3  | Piccolo | Namekiano          | 5000        |
| 4  | Gohan   | Saiyan             | 6000        |
| 5  | Frieza  | Emperador del Mal  | 10000       |
| 6  | Cell    | Bio-Androide       | 8000        |
| 7  | Trunks  | Saiyan             | 7500        |
| 8  | Krillin | Humano             | 3000        |
| 9  | Goten   | Saiyan             | 4000        |
| 10 | Bulma   | Humano             | 2000        |
+----+---------+--------------------+-------------+
```

1. **COUNT()**: Contar el número total de guerreros.

```
SELECT COUNT(*) AS total_guerreros FROM Guerreros;
```

Resultado:

```
+-----------------+
| total_guerreros |
+-----------------+
| 10              |
+-----------------+
```

2. **SUM()**: Calcular la suma total de los niveles de poder de todos los guerreros.

```
SELECT SUM(nivel_poder) AS suma_niveles_poder FROM Guerreros;
```

Resultado:

```
+--------------------+
| suma_niveles_poder  |
+--------------------+
| 62500               |
+--------------------+
```

3. **AVG()**: Calcular el promedio de los niveles de poder de todos los guerreros.

```
SELECT AVG(nivel_poder) AS promedio_niveles_poder FROM Guerreros;
```

Resultado:

```
+-------------------------+
| promedio_niveles_poder  |
+-------------------------+
| 6250.000                |
+-------------------------+
```

4. **MIN()**: Encontrar el nivel de poder mínimo.

```
SELECT MIN(nivel_poder) AS nivel_poder_minimo FROM Guerreros;
```

Resultado:

```
+-------------------+
| nivel_poder_minimo|
+-------------------+
| 2000              |
+-------------------+
```

5. **MAX()**: Encontrar el guerrero con el nivel de poder máximo.

```
SELECT MAX(nivel_poder) AS nivel_poder_maximo FROM Guerreros;
```

Resultado:

```
+-------------------+
| nivel_poder_maximo|
+-------------------+
| 10000             |
+-------------------+
```

Más sobre las consultas de agregación de datos (debes tener la bbdd sakila para seguir las instrucciones)**:**

-

¿Por qué da error y no se puede realizar la siguiente consulta?:

```
SELECT nombre, MAX(nivel_poder) AS nivel_poder_maximo FROM Guerreros;
```

El error se produce porque en la consulta se está utilizando una función de agregación (`MAX()`) en la columna `nivel_poder`, pero también estás seleccionando la columna `nombre`, que no está siendo agregada.

Cuando usas una función de agregación como `MAX()`, el motor de base de datos agrupa los registros según los criterios especificados en la consulta y realiza una operación de agregación (en este caso, encontrar el valor máximo de `nivel_poder`). Sin embargo, la columna `nombre` no está siendo agregada ni está incluida en la operación de agregación.

El motor de base de datos no sabe qué hacer con la columna `nombre` porque no está incluida en la operación de agregación y no está agrupada por ninguna función de agrupación como `GROUP BY`. Por lo tanto, muestra un error para indicar que la consulta no es válida según la configuración actual del servidor MySQL, que requiere que todas las columnas en la lista de selección sean agregadas o estén incluidas en una cláusula `GROUP BY`.

Para corregir este error, **se usan las subconsultas** para obtener el nombre del guerrero con el nivel de poder máximo, o simplemente no seleccionar la columna `nombre` en la misma consulta que utiliza la función de agregación.

5.3. Subconsultas

Una subconsulta es una "consulta dentro de otra consulta". Imagina que estás buscando información en un libro, pero antes necesitas consultar otro libro para encontrar la información que necesitas. La consulta externa utiliza los resultados de la consulta interna para tomar decisiones o filtrar resultados.

La sintaxis básica de una subconsulta es la siguiente:

```
SELECT columna
FROM tabla
WHERE condicion IN (SELECT columna FROM otra_tabla WHERE otra_condicion);
```

Donde cada parte de la sintaxis hace:

- `SELECT columna`: Especifica la columna que quieres seleccionar de la tabla principal.
- `FROM tabla`: Indica la tabla principal de la que se seleccionarán los datos.
- `WHERE condicion`: Define la condición que deben cumplir las filas de la tabla principal para ser seleccionadas.
- `IN (SELECT columna FROM otra_tabla WHERE otra_condicion)`: Esta es la subconsulta. Se ejecuta primero y devuelve un conjunto de resultados que se utilizan como parte de la condición de la consulta principal. Puede contener su propia cláusula `SELECT`, `FROM` y `WHERE` para filtrar los datos (el IN no es obligatorio, puede ser un operador relacional, etc).

En el siguiente ejemplo, queremos encontrar guerreros cuyo nivel de poder sea inferior al de Vegeta. Podríamos escribir la consulta así:

```
-- Obtiene los guerreros cuyo nivel de poder sean inferiores al de Vegeta.
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder < (
    SELECT nivel_poder
    FROM Guerreros
    WHERE nombre = 'Vegeta'
);

+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Piccolo |        5000 |
+---------+-------------+
```

En esta consulta, la subconsulta `(SELECT nivel_poder FROM Guerreros WHERE nombre = 'Vegeta')` obtiene el nivel de poder de Vegeta. Luego, la consulta externa selecciona los nombres y niveles de poder de los guerreros cuyo nivel de poder es menor que el de Vegeta.

Así es como funcionan las subconsultas: la consulta externa utiliza los resultados de la consulta interna para tomar decisiones o realizar operaciones más avanzadas.

También puedes anidar subconsultas tantas veces como sea necesario. Por ejemplo:

```
-- Obtén el nombre y el nivel de poder de los guerreros que tengan más poder que Vegeta y menos que Goku.
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder > (
SELECT nivel_poder
    FROM Guerreros
    WHERE nombre = 'Vegeta')
AND nivel_poder < (
SELECT nivel_poder
    FROM Guerreros
    WHERE nombre = 'Goku');

Empty set (0,00 sec)
```

Esta consulta obtiene los guerreros cuyo nivel de poder está entre el de Vegeta y el de Goku. Si no hay guerreros en medio, el resultado será un "conjunto vacío" o "Empty set".

Es importante tener en cuenta que las subconsultas que utilizan operadores de comparación como `>`, `<`, `>=`, etc., deben devolver un único valor. De lo contrario, se producirá un error.

A veces, necesitamos comparar un valor con múltiples valores devueltos por una subconsulta. Para esto, podemos utilizar instrucciones especiales entre el operador y la subconsulta:

- `ANY`: Compara un valor con cualquier valor devuelto por la subconsulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta.
- `ALL`: Compara un valor con todos los valores devueltos por la subconsulta. La instrucción resulta cierta si es cierta toda la comparación con los registros de la subconsulta.
- `IN`: Comprueba si un valor está presente en un conjunto de valores devueltos por la subconsulta. No usa comparador, ya que sirve para comprobar si un valor se encuentra en el resultado de la subconsulta.
- `NOT IN`: Comprueba si un valor no está presente en un conjunto de valores devueltos por la subconsulta. Si no está presente, devuelve true. Comprueba si un valor no se encuentra en una subconsulta.

Por ejemplo para obtener el guerrero con mayor nivel de poder:

```
SELECT nombre, nivel_poder
FROM Guerreros
WHERE nivel_poder >= ALL (
    SELECT nivel_poder
    FROM Guerreros
);
```

Esta consulta seleccionará el guerrero cuyo nivel de poder sea mayor o igual que todos los demás niveles de poder en la tabla.

```
+---------+-------------+
| nombre  | nivel_poder |
+---------+-------------+
| Goku    |        9000 |
+---------+-------------+
```

5.4. Consultas multitabla. Composición Interna

Las consultas multitabla nos permiten obtener información de más de una tabla en una sola consulta. La principal diferencia con las consultas simples es que debemos especificar qué tablas vamos a utilizar y cómo están relacionadas entre sí en la cláusula `FROM`.

Para realizar este tipo de consultas, existen dos enfoques principales. El primero implica obtener el producto cartesiano de las tablas (es decir, todas las combinaciones posibles de filas entre las tablas) y luego aplicar un filtro para relacionar los datos que tienen en común. El segundo enfoque, utiliza todas las cláusulas de tipo JOIN para establecer las relaciones entre las tablas de una manera más específica y flexible.

En este apartado de teoría se trabajará sobre las siguientes tablas:

Tabla `guerreros`:

```
+-------------+------------+----------+------------+
| id_guerrero | nombre     | raza     | nivel_poder|
+-------------+------------+----------+------------+
| 1           | Goku       | Saiyan   | 9000       |
| 2           | Vegeta     | Saiyan   | 8500       |
| 3           | Piccolo    | Namekian | 5000       |
| 4           | Gohan      | Saiyan   | 6000       |
| 5           | Krilin     | Humano   | 4000       |
+-------------+------------+----------+------------+
```

Tabla `tecnicas`:

```
+-------------+-------------+----------------------+
| id_tecnica  | id_guerrero | nombre_tecnica       |
+-------------+-------------+----------------------+
| 1           | 1           | Kamehameha           |
| 2           | 1           | Genki Dama           |
| 3           | 2           | Final Flash          |
| 4           | 3           | Makankosappo         |
| 5           | 4           | Masenko              |
| 6           | 5           | Kienzan              |
+-------------+-------------+----------------------+
```

Composición Cruzada (Producto Cartesiano): En este tipo de operación se obtiene un conjunto de elementos que representa todas las posibles combinaciones entre los elementos de dos conjuntos. En el ejemplo proporcionado, se busca obtener todas las combinaciones posibles entre los guerreros y las técnicas. Esto se logra mediante la especificación de ambas tablas en la cláusula FROM, sin ningún tipo de relación explícita entre ellas:

```
SELECT *
FROM guerreros, tecnicas;
```

El resultado de esta consulta mostrará todas las combinaciones posibles entre los guerreros y las técnicas, incluso si no hay una relación específica entre ellas. Esto puede resultar en un conjunto de resultados muy grande y no siempre útil, por lo que generalmente se prefiere utilizar la composición interna cuando se busca obtener datos relacionados entre múltiples tablas.

```
+-------------+---------+----------+-------------+------------+-------------+----------------+
| id_guerrero | nombre  | raza     | nivel_poder | id_tecnica | id_guerrero | nombre_tecnica |
+-------------+---------+----------+-------------+------------+-------------+----------------+
|           5 | Krilin  | Humano   |        4000 |          1 |           1 | Kamehameha     |
|           4 | Gohan   | Saiyan   |        6000 |          1 |           1 | Kamehameha     |
|           3 | Piccolo | Namekian |        5000 |          1 |           1 | Kamehameha     |
|           2 | Vegeta  | Saiyan   |        8500 |          1 |           1 | Kamehameha     |
|           1 | Goku    | Saiyan   |        9000 |          1 |           1 | Kamehameha     |
|           5 | Krilin  | Humano   |        4000 |          2 |           1 | Genki Dama     |
|           4 | Gohan   | Saiyan   |        6000 |          2 |           1 | Genki Dama     |
|           3 | Piccolo | Namekian |        5000 |          2 |           1 | Genki Dama     |
|           2 | Vegeta  | Saiyan   |        8500 |          2 |           1 | Genki Dama     |
|           1 | Goku    | Saiyan   |        9000 |          2 |           1 | Genki Dama     |
|           5 | Krilin  | Humano   |        4000 |          3 |           2 | Final Flash    |
|           4 | Gohan   | Saiyan   |        6000 |          3 |           2 | Final Flash    |
|           3 | Piccolo | Namekian |        5000 |          3 |           2 | Final Flash    |
|           2 | Vegeta  | Saiyan   |        8500 |          3 |           2 | Final Flash    |
|           1 | Goku    | Saiyan   |        9000 |          3 |           2 | Final Flash    |
|           5 | Krilin  | Humano   |        4000 |          4 |           3 | Makankosappo   |
|           4 | Gohan   | Saiyan   |        6000 |          4 |           3 | Makankosappo   |
|           3 | Piccolo | Namekian |        5000 |          4 |           3 | Makankosappo   |
|           2 | Vegeta  | Saiyan   |        8500 |          4 |           3 | Makankosappo   |
|           1 | Goku    | Saiyan   |        9000 |          4 |           3 | Makankosappo   |
|           5 | Krilin  | Humano   |        4000 |          5 |           4 | Masenko        |
|           4 | Gohan   | Saiyan   |        6000 |          5 |           4 | Masenko        |
|           3 | Piccolo | Namekian |        5000 |          5 |           4 | Masenko        |
|           2 | Vegeta  | Saiyan   |        8500 |          5 |           4 | Masenko        |
|           1 | Goku    | Saiyan   |        9000 |          5 |           4 | Masenko        |
|           5 | Krilin  | Humano   |        4000 |          6 |           5 | Kienzan        |
|           4 | Gohan   | Saiyan   |        6000 |          6 |           5 | Kienzan        |
|           3 | Piccolo | Namekian |        5000 |          6 |           5 | Kienzan        |
|           2 | Vegeta  | Saiyan   |        8500 |          6 |           5 | Kienzan        |
|           1 | Goku    | Saiyan   |        9000 |          6 |           5 | Kienzan        |
+-------------+---------+----------+-------------+------------+-------------+----------------+
```

Composición Interna (Intersección): Ahora bien, teniendo en cuenta que la operación anterior a priori no tiene mucho sentido; es posible adquirir el sentido si ésta se filtra para obtener solo los elementos comunes entre dos conjuntos. En el ejemplo de las tablas proporcionadas, sería lógico obtener un listado de guerreros y sus técnicas reales. Para lograrlo, se utiliza la cláusula WHERE para indicar la relación entre las dos tablas (guerrero y técnicas) a través de una columna compartida (en este caso, `id_guerrero` en ambas tablas).

De esta forma si queremos obtener un listado de guerreros y las técnicas que poseen, podríamos realizar una consulta similar a la siguiente:

```
SELECT *
FROM guerreros, tecnicas
WHERE guerreros.id_guerrero = tecnicas.id_guerrero;
```

El resultado de esta consulta mostrará solo las filas donde haya una coincidencia en el `id_guerrero` entre ambas tablas, es decir, solo mostrará las técnicas que pertenecen a cada guerrero.

```
+-------------+-----------+----------+-----------+------------+-------------+----------------------+
| id_guerrero | nombre    | raza     | nivel_poder| id_tecnica | id_guerrero | nombre_tecnica       |
+-------------+-----------+----------+-----------+------------+-------------+----------------------+
| 1           | Goku      | Saiyan   | 9000      | 1          | 1           | Kamehameha           |
| 1           | Goku      | Saiyan   | 9000      | 2          | 1           | Genki Dama           |
| 2           | Vegeta    | Saiyan   | 8500      | 3          | 2           | Final Flash          |
| 3           | Piccolo   | Namekian | 5000      | 4          | 3           | Makankosappo         |
| 4           | Gohan     | Saiyan   | 6000      | 5          | 4           | Masenko              |
| 5           | Krilin    | Humano   | 4000      | 6          | 5           | Kienzan              |
+-------------+-----------+----------+-----------+------------+-------------+----------------------+
```

Tenga en cuenta que con la **operación de intersección** sólo obtendremos los elementos que existan en ambos conjuntos. Por lo tanto, en el ejemplo anterior puede ser que existan filas en la tabla `tecnicas` que no aparecen en el resultado porque no tienen ningún guerrero asociado, y viceversa.

Composición Interna (INNER JOIN O JOIN): Para realizar la composición interna, MySQL tiene una cláusula específica (JOIN o INNER JOIN -el uso de la palabra reservada INNER es opcional-)para facilitar en el futuro las consultas más complejas. La operación de intersección utilizando JOIN igualmente se utiliza para combinar las filas de ambas tablas basadas en una condición específica. Manteniendo el ejemplo anterior, la condición sería la igualdad de los valores en la columna "id_guerrero".

```
SELECT *
FROM guerreros
JOIN tecnicas
ON guerreros.id_guerrero = tecnicas.id_guerrero;
```

Esta consulta devuelve el mismo resultado que la consulta anterior, mostrando solo las filas donde haya una coincidencia en el "id_guerrero" entre ambas tablas, es decir, las técnicas que pertenecen a cada guerrero.

```
+-------------+------------+----------+------------+-------------+-------------+----------------------+
| id_guerrero | nombre     | raza     | nivel_poder| id_tecnica  | id_guerrero | nombre_tecnica       |
+-------------+------------+----------+------------+-------------+-------------+----------------------+
| 1           | Goku       | Saiyan   | 9000       | 1           | 1           | Kamehameha           |
| 1           | Goku       | Saiyan   | 9000       | 2           | 1           | Genki Dama           |
| 2           | Vegeta     | Saiyan   | 8500       | 3           | 2           | Final Flash          |
| 3           | Piccolo    | Namekian | 5000       | 4           | 3           | Makankosappo         |
| 4           | Gohan      | Saiyan   | 6000       | 5           | 4           | Masenko              |
| 5           | Krilin     | Humano   | 4000       | 6           | 5           | Kienzan              |
+-------------+------------+----------+------------+-------------+-------------+----------------------+
```

Esta consulta es equivalente a la anterior en términos de funcionalidad, pero utiliza la sintaxis JOIN para realizar la operación en lugar de la lista de tablas separadas por comas con la cláusula WHERE para la condición de unión.

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 `guerrero` y `tecnicas`.

```
SELECT *
FROM guerreros
JOIN tecnicas;
```

```
+-------------+---------+----------+-------------+------------+-------------+----------------+
| id_guerrero | nombre  | raza     | nivel_poder | id_tecnica | id_guerrero | nombre_tecnica |
+-------------+---------+----------+-------------+------------+-------------+----------------+
|           5 | Krilin  | Humano   |        4000 |          1 |           1 | Kamehameha     |
|           4 | Gohan   | Saiyan   |        6000 |          1 |           1 | Kamehameha     |
|           3 | Piccolo | Namekian |        5000 |          1 |           1 | Kamehameha     |
|           2 | Vegeta  | Saiyan   |        8500 |          1 |           1 | Kamehameha     |
|           1 | Goku    | Saiyan   |        9000 |          1 |           1 | Kamehameha     |
|           5 | Krilin  | Humano   |        4000 |          2 |           1 | Genki Dama     |
|           4 | Gohan   | Saiyan   |        6000 |          2 |           1 | Genki Dama     |
|           3 | Piccolo | Namekian |        5000 |          2 |           1 | Genki Dama     |
|           2 | Vegeta  | Saiyan   |        8500 |          2 |           1 | Genki Dama     |
|           1 | Goku    | Saiyan   |        9000 |          2 |           1 | Genki Dama     |
|           5 | Krilin  | Humano   |        4000 |          3 |           2 | Final Flash    |
|           4 | Gohan   | Saiyan   |        6000 |          3 |           2 | Final Flash    |
|           3 | Piccolo | Namekian |        5000 |          3 |           2 | Final Flash    |
|           2 | Vegeta  | Saiyan   |        8500 |          3 |           2 | Final Flash    |
|           1 | Goku    | Saiyan   |        9000 |          3 |           2 | Final Flash    |
|           5 | Krilin  | Humano   |        4000 |          4 |           3 | Makankosappo   |
|           4 | Gohan   | Saiyan   |        6000 |          4 |           3 | Makankosappo   |
|           3 | Piccolo | Namekian |        5000 |          4 |           3 | Makankosappo   |
|           2 | Vegeta  | Saiyan   |        8500 |          4 |           3 | Makankosappo   |
|           1 | Goku    | Saiyan   |        9000 |          4 |           3 | Makankosappo   |
|           5 | Krilin  | Humano   |        4000 |          5 |           4 | Masenko        |
|           4 | Gohan   | Saiyan   |        6000 |          5 |           4 | Masenko        |
|           3 | Piccolo | Namekian |        5000 |          5 |           4 | Masenko        |
|           2 | Vegeta  | Saiyan   |        8500 |          5 |           4 | Masenko        |
|           1 | Goku    | Saiyan   |        9000 |          5 |           4 | Masenko        |
|           5 | Krilin  | Humano   |        4000 |          6 |           5 | Kienzan        |
|           4 | Gohan   | Saiyan   |        6000 |          6 |           5 | Kienzan        |
|           3 | Piccolo | Namekian |        5000 |          6 |           5 | Kienzan        |
|           2 | Vegeta  | Saiyan   |        8500 |          6 |           5 | Kienzan        |
|           1 | Goku    | Saiyan   |        9000 |          6 |           5 | Kienzan        |
+-------------+---------+----------+-------------+------------+-------------+----------------+
```

2 Errores comunes

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

Consulta **incorrecta**

```
SELECT *
FROM guerreros, tecnicas
WHERE tecnicas.nombre_tecnica = 'Kamehameha';
```

Consulta **correcta**

```
SELECT *
FROM guerreros, tecnicas
WHERE guerreros.id_guerrero = tecnicas.id_guerrero AND tecnicas.nombre_tecnica = 'Kamehameha';
```

2. Nos olvidamos de incluir `ON` en las consultas de tipo `INNER JOIN`.
Consulta **incorrecta**

```
SELECT *
FROM guerreros JOIN tecnicas
WHERE tecnicas.nombre_tecnica = 'Kamehameha';
```

Consulta **correcta**

```
SELECT *
FROM guerreros JOIN tecnicas
ON guerreros.id_guerrero = tecnicas.id_guerrero
WHERE tecnicas.nombre_tecnica = 'Kamehameha';
```

3. Relacionamos las tablas utilizando nombres de columnas incorrectos.
Consulta **incorrecta**

```
SELECT *
FROM producto INNER JOIN fabricante
ON guerreros.id = tecnicas.id_tecnica;
```

Consulta **correcta**

```
SELECT *
FROM producto INNER JOIN fabricante
ON guerreros.id_guerrero = tecnicas.id_guerrero;
```

4**. Cuando hacemos la intersección de tres tablas con `INNER JOIN` nos olvidamos de incluir `ON` en alguna de las intersecciones (imagina una tercera tabla 'planetas' y un correspondiente campo 'id_planeta' en guerreros).**
Consulta **incorrecta**

```
SELECT DISTINCT nombre, raza, nombre_planeta
FROM guerreros JOIN tecnicas
JOIN planetas
ON guerreros.id_guerrero = tecnicas.id_guerrero;
```

Consulta **correcta**

```
SELECT DISTINCT nombre, raza, nombre_planeta
FROM guerreros JOIN tecnicas
ON guerreros.id_guerrero = tecnicas.id_guerrero;
JOIN pago
ON guerreros.id_planeta = planetas.id_planeta;
```

Ejemplo resuelto de la relación de ejercicios de la tienda de informática. Con consultas multitabla correlacionadas.
Teniendo la base de datos que contiene las dos siguientes tablas, **lista el nombre de cada fabricante con el nombre y el precio de su producto más caro.**

```
mysql> select * from fabricante;
```

```
+----+-----------------+
| id | nombre          |
+----+-----------------+
|  1 | Asus            |
|  2 | Lenovo          |
|  3 | Hewlett-Packard |
|  4 | Samsung         |
|  5 | Seagate         |
|  6 | Crucial         |
|  7 | Gigabyte        |
|  8 | Huawei          |
|  9 | Xiaomi          |
+----+-----------------+
9 rows in set (0,00 sec)

mysql> select * from producto;
+----+---------------------------------+--------+---------------+
| id | nombre                          | precio | id_fabricante |
+----+---------------------------------+--------+---------------+
|  1 | Disco duro SATA3 1TB            |  86.99 |             5 |
|  2 | Memoria RAM DDR4 8GB            |    120 |             6 |
|  3 | Disco SSD 1 TB                  | 150.99 |             4 |
|  4 | GeForce GTX 1050Ti              |    185 |             7 |
|  5 | GeForce GTX 1080 Xtreme         |    755 |             6 |
|  6 | Monitor 24 LED Full HD          |    202 |             1 |
|  7 | Monitor 27 LED Full HD          | 245.99 |             1 |
|  8 | Portátil Yoga 520               |    559 |             2 |
|  9 | Portátil Ideapd 320             |    444 |             2 |
| 10 | Impresora HP Deskjet 3720       |  59.99 |             3 |
| 11 | Impresora HP Laserjet Pro M26nw |    180 |             3 |
+----+---------------------------------+--------+---------------+
11 rows in set (0.00 sec)
```

Lo normal en estos casos es proceder a resolver paso a paso, tal y como lo haría la mente humana (sin usar MySQL).

Teniendo eso en cuenta, el primer paso que haría una mente humana debería ser obtener de la tabla producto el precio más caro que tiene cada fabricante, para ello, habría que hacer:

```
mysql> SELECT id_fabricante, MAX(precio) AS precio_maximo FROM producto GROUP BY id_fabricante;
+---------------+---------------+
| id_fabricante | precio_maximo |
+---------------+---------------+
|             1 |        245.99 |
|             2 |           559 |
|             3 |           180 |
|             4 |        150.99 |
|             5 |         86.99 |
|             6 |           755 |
|             7 |           185 |
+---------------+---------------+
7 rows in set (0.00 sec)
```

A esta tabla virtual le vamos a dar el nombre de ‘t3’.

**PREGUNTA 1.- ¿Si se está trabajando como lo haría la mente humana, por qué no se muestra directamente en esta consulta también el nombre del producto al que corresponde el precio más caro?. ¿Qué sucede al hacer esto?¿por qué está MAL?.

```
mysql> SELECT id_fabricante, nombre, MAX(precio)
```

```
    -> FROM producto
    -> GROUP BY id_fabricante, nombre;
+---------------+--------------------------------+-------------+
| id_fabricante | nombre                         | MAX(precio) |
+---------------+--------------------------------+-------------+
|             5 | Disco duro SATA3 1TB           |       86.99 |
|             6 | Memoria RAM DDR4 8GB           |         120 |
|             4 | Disco SSD 1 TB                 |      150.99 |
|             7 | GeForce GTX 1050Ti             |         185 |
|             6 | GeForce GTX 1080 Xtreme        |         755 |
|             1 | Monitor 24 LED Full HD         |         202 |
|             1 | Monitor 27 LED Full HD         |      245.99 |
|             2 | Portátil Yoga 520              |         559 |
|             2 | Portátil Ideapd 320            |         444 |
|             3 | Impresora HP Deskjet 3720      |       59.99 |
|             3 | Impresora HP Laserjet Pro M26nw |         180 |
+---------------+--------------------------------+-------------+
11 rows in set (0.00 sec)
```

Está mal porque si se agrupa por id_fabricante + nombre, se van a buscar los precios máximos para cada combinación id_fabricante+nombre, y como la unión de esos dos campos no tienen repeticiones, se muestra de nuevo todos los registros (la tabla producto tal cual).

Para mostrar correctamente el nombre del artículo junto con el id de fabricante y su precio máximo, habría que unir la tabla virtual t3 (que proviene de la tabla producto) con la propia tabla producto. De ésta forma se podría obtener la columna correspondiente al nombre del producto.

**PREGUNTA 2.- Al unir ambas tablas, ¿qué campo se debería igualar para obtener como resultado sólo los registros de precio máximo?

```
mysql> SELECT t3.id_fabricante, p.nombre, t3.precio_maximo
    -> FROM producto p
    -> JOIN (SELECT id_fabricante, MAX(precio) AS precio_maximo FROM producto GROUP BY id_fabricante) AS t3
    -> ON t3.precio_maximo = p.precio;
+---------------+--------------------------------+---------------+
| id_fabricante | nombre                         | precio_maximo |
+---------------+--------------------------------+---------------+
|             5 | Disco duro SATA3 1TB           |         86.99 |
|             4 | Disco SSD 1 TB                 |        150.99 |
|             7 | GeForce GTX 1050Ti             |           185 |
|             6 | GeForce GTX 1080 Xtreme        |           755 |
|             1 | Monitor 27 LED Full HD         |        245.99 |
|             2 | Portátil Yoga 520              |           559 |
|             3 | Impresora HP Laserjet Pro M26nw |           180 |
+---------------+--------------------------------+---------------+
7 rows in set (0.00 sec)
```

De nuevo, al igual que lo haría el cerebro humano, se llega a la conclusión de que hay que igualar por los campos precios de ambas tablas.

Si se reflexiona sobre la consulta anterior, en la línea del SELECT, se puede prescindir de los campos t3 y se puede hacer referencia a ellos pero desde la tabla producto, ya que son el mismo campo, y queda de forma más inteligible.

```
mysql> SELECT p.id_fabricante, p.nombre, p.precio_maximo        -> FROM producto p
    -> JOIN (SELECT id_fabricante, MAX(precio) AS precio_maximo FROM producto GROUP BY id_fabricante) AS t3
    -> ON t3.precio_maximo = p.precio;
+---------------+--------------------------------+---------------+
| id_fabricante | nombre                         | precio_maximo |
+---------------+--------------------------------+---------------+
|             5 | Disco duro SATA3 1TB           |         86.99 |
|             4 | Disco SSD 1 TB                 |        150.99 |
|             7 | GeForce GTX 1050Ti             |           185 |
|             6 | GeForce GTX 1080 Xtreme        |           755 |
|             1 | Monitor 27 LED Full HD         |        245.99 |
|             2 | Portátil Yoga 520              |           559 |
|             3 | Impresora HP Laserjet Pro M26nw |           180 |
+---------------+--------------------------------+---------------+
7 rows in set (0.00 sec)
```

Por último, ya solo queda mostrar el nombre del fabricante, en lugar del id del fabricante, y esta información sólo puede obtenerse de la tabla fabricante. Por esto mismo, habrá que unir y relacionar la tabla fabricante con este resultado obtenido anteriormente (que ya proviene de hacer una unión previa).

Si se reflexiona, la forma más lógica de hacerlo pues, es realizando un doble JOIN, siguiendo el orden del párrafo anterior.

```
mysql> SELECT
    ->     f.nombre AS nombre_fabricante,
    ->     p.nombre AS nombre_producto,
    ->     p.precio AS precio_máximo
    -> FROM
    ->    fabricante f
    -> JOIN
    ->    producto p
    -> ON f.id = p.id_fabricante
    -> JOIN
    -> (SELECT
    ->         id_fabricante,
    ->         MAX(precio) AS precio_maximo
    ->     FROM producto
    ->     GROUP BY id_fabricante) AS t3
    -> ON t3.precio_maximo = p.precio;
```

```
+------------------+--------------------------------+----------------+
| nombre_fabricante | nombre_producto                | precio_máximo  |
+------------------+--------------------------------+----------------+
| Seagate           | Disco duro SATA3 1TB           |          86.99 |
| Samsung           | Disco SSD 1 TB                 |         150.99 |
| Gigabyte          | GeForce GTX 1050Ti             |            185 |
| Crucial           | GeForce GTX 1080 Xtreme        |            755 |
| Asus              | Monitor 27 LED Full HD         |         245.99 |
| Lenovo            | Portátil Yoga 520              |            559 |
| Hewlett-Packard   | Impresora HP Laserjet Pro M26nw |            180 |
+------------------+--------------------------------+----------------+
7 rows in set (0.00 sec)
```

Consultas sobre varias tablas. Composición externa

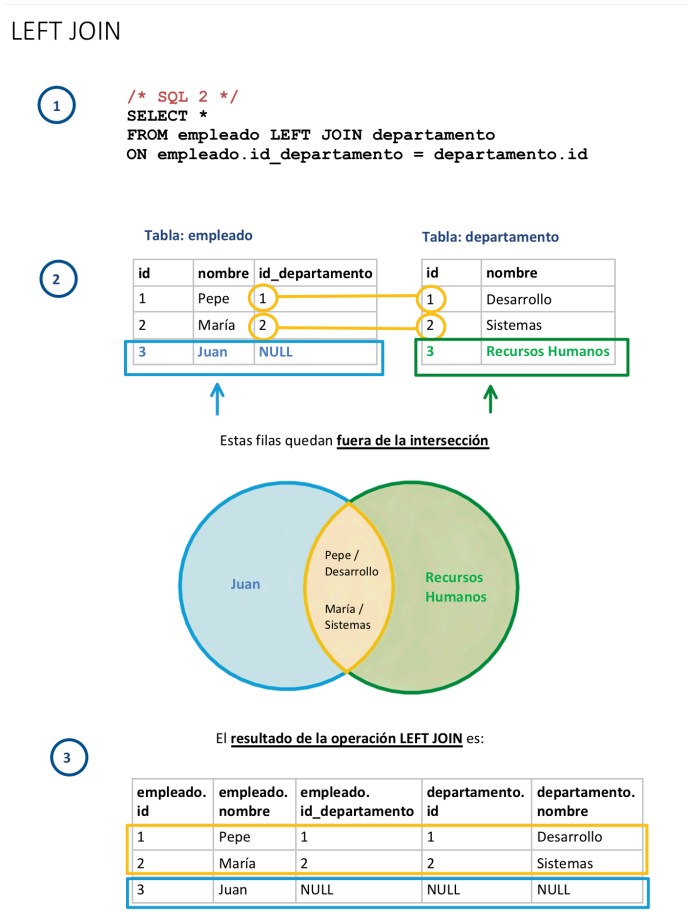
1 Composiciones externas

- Join externa
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `NATURAL LEFT OUTER JOIN`
  - `NATURAL RIGHT OUTER JOIN`

Ejemplo de `LEFT OUTER JOIN`:

```
SELECT *
FROM empleado LEFT JOIN departamento
ON empleado.id_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`.



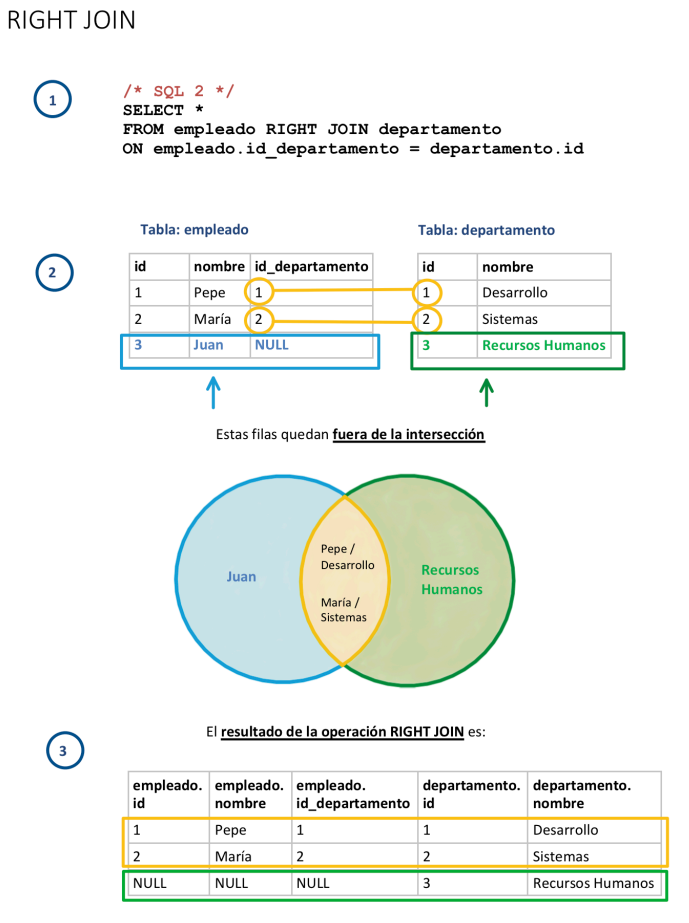

```
SELECT *
FROM empleado
LEFT JOIN departamento
ON empleado.id_departamento = departamento.id;
```

```
+----+--------+--------------------+------+------------+
| id | nombre | id_departamento    | id   | nombre     |
+----+--------+--------------------+------+------------+
|  1 | Pepe   |                  1 |    1 | Desarrollo |
|  2 | María  |                  2 |    2 | Sistemas   |
|  3 | Juan   |               NULL | NULL | NULL       |
+----+--------+--------------------+------+------------+
```

Ejemplo de `RIGHT OUTER JOIN`:

```
SELECT *
FROM empleado RIGHT JOIN departamento
ON empleado.id_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`.




```
SELECT *
FROM departamento
RIGHT JOIN empleado
ON departamento.id = empleado.id_departamento;
```

```
+------+------------+----+--------+--------------------+
| id   | nombre     | id | nombre | id_departamento    |
+------+------------+----+--------+--------------------+
|    1 | Desarrollo |  1 | Pepe   |                  1 |
|    2 | Sistemas   |  2 | María  |                  2 |
| NULL | NULL       |  3 | Juan   |               NULL |
+------+------------+----+--------+--------------------+
```

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 union 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.id_departamento = departamento.id

UNION

SELECT *
FROM empleado RIGHT JOIN departamento
ON empleado.id_departamento = departamento.id;
```

```
+------+--------+--------------------+------+------------------+
| id   | nombre | id_departamento    | id   | nombre           |
+------+--------+--------------------+------+------------------+
|    1 | Pepe   |                  1 |    1 | Desarrollo       |
|    2 | María  |                  2 |    2 | Sistemas         |
|    3 | Juan   |               NULL | NULL | NULL             |
| NULL | NULL   |               NULL |    3 | Recursos Humanos |
+------+--------+--------------------+------+------------------+
```

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`.

Ejemplo:

```
/* Solución 1 */
SELECT departamento.nombre, empleado.nombre
FROM empleado LEFT JOIN departamento
ON empleado.id_departamento=departamento.id

UNION

SELECT departamento.nombre, empleado.nombre
FROM empleado RIGHT JOIN departamento
ON empleado.id_departamento=departamento.id
ORDER BY 1, 2;
```

```
+------------------+--------+
| nombre           | nombre |
+------------------+--------+
| NULL             | Juan   |
| Desarrollo       | Pepe   |
| Recursos Humanos | NULL   |
| Sistemas         | María  |
+------------------+--------+
```

```
/* Solución 2 */
SELECT
  departamento.nombre AS nombre_departamento,
  empleado.nombre
FROM empleado LEFT JOIN departamento
ON empleado.id_departamento=departamento.id

UNION

SELECT departamento.nombre, empleado.nombre
FROM empleado RIGHT JOIN departamento
ON empleado.id_departamento=departamento.id
ORDER BY nombre_departamento;
```

```
+--------------------+--------+
| nombre_departamento | nombre |
+--------------------+--------+
| NULL               | Juan   |
| Desarrollo         | Pepe   |
| Recursos Humanos   | NULL   |
| Sistemas           | María  |
+--------------------+--------+
```

Ejemplo de `NATURAL LEFT JOIN`:

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

Resultado:

```
+----+--------+--------------------+
| id | nombre | id_departamento    |
+----+--------+--------------------+
|  1 | Pepe   |                  1 |
|  2 | María  |                  2 |
|  3 | Juan   |               NULL |
+----+--------+--------------------+
```

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.

2 Errores comunes

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;
```