Sentencias SELECT y WHERE en MySQL

En el desarrollo de sistemas informáticos, la gestión y manipulación de datos es fundamental. MySQL, como sistema de gestión de bases de datos relacional, utiliza el lenguaje SQL (Structured Query Language) para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Dos de las sentencias más utilizadas son:

- SELECT: Para seleccionar y mostrar datos.
- WHERE: Para filtrar registros según condiciones específicas.

Objetivos

- Entender la estructura básica de una consulta SQL con `SELECT` y `WHERE`.
- Aprender a filtrar datos usando operadores lógicos y comparativos.
- Aplicar ejemplos prácticos en escenarios reales.

Sentencia SELECT

Sintaxis Básica

sql
SELECT [columnas] FROM [nombre_tabla];

- [columnas]: Pueden ser nombres específicos (ej. `nombre, edad`) o `*` para seleccionar todas las columnas.
- [nombre_tabla]: Nombre de la tabla desde la que se extraen los datos.

Operadores Aritméticos

- **Suma:**
  Forma 1
  sql
  SELECT [columna_1] + [columna_2] FROM [nombre_tabla];

  Forma 2
  sql
  *SELECT * FROM [nombre_tabla] WHERE [columna_1] + [columna_2] = Valor;*

  Ejemplo:
  Sql
  *SELECT * FROM programs where valor_1 + valor_2 = 5;*

- **Resta:**
  Forma 1
  sql
  SELECT [columna_1] - [columna_2] FROM [nombre_tabla];

  Forma 2
  sql
  *SELECT * FROM [nombre_tabla] WHERE [columna_1] - [columna_2] = [Valor];*

- **Multiplicación:**
  Forma 1
  sql
  *SELECT [columna_1] * [columna_2] FROM [nombre_tabla];*

  Forma 2
  *SELECT [columna_1] * [Valor] FROM [nombre_tabla];*

  Forma 3
  *SELECT * FROM [nombre_tabla] WHERE [columna_1] * [Valor1] = [Valor2]*

- **División:**
  Forma 1
  sql
  SELECT [columna_1] / [columna_2] FROM [nombre_tabla];

  Forma 2
  *SELECT [columna_1] / [columna_2] * [Valor] FROM [nombre_tabla];*

  Forma 3
  *SELECT * FROM [nombre_tabla] WHERE [columna_1] / [Valor1] >= [Valor2];*

- **Modulo:**
  Forma 1
  sql
  SELECT [columna_1] % [columna_2] FROM [nombre_tabla];

  Forma 2
  *SELECT * FROM [nombre_tabla] WHERE [columna_1] % [Valor1] >= [Valor2];*

Ejemplos

Ejemplo 1: Seleccionar todas las columnas

Sql
```
CREATE TABLE empleados (
    empleado_id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50),
    apellidos VARCHAR(100),
    departamento ENUM('ventas', 'comercial', 'gerencia'),
    salario DECIMAL(10, 2)
);
```

Descripción de los campos:

- empleado_id: número entero que se incrementa automáticamente.
- nombre: cadena de texto de hasta 50 caracteres.
- apellidos: cadena de texto de hasta 100 caracteres.
- departamento: tipo ENUM que solo permite los valores especificados (ventas, comercial, gerencia).
- salario: número decimal con hasta 10 dígitos y 2 decimales.

Sql
```
INSERT INTO empleados (nombre, apellidos, departamento, salario) VALUES
        ('Ana', 'López Pérez', 'ventas', 3200.50),
        ('Carlos', 'Mendoza Ruiz', 'ventas', 2850.75),
        ('María', 'García Torres', 'comercial', 3100.00),
        ('Javier', 'Romero Sánchez', 'comercial', 2950.25),
        ('Lucía', 'Hernández Gómez', 'gerencia', 4500.00),
        ('Diego', 'Castro Ortega', 'gerencia', 4800.00);

SELECT * FROM empleados;
```

Muestra todos los registros de la tabla `empleados`.

Ejemplo 2: Seleccionar columnas específicas

sql
```
SELECT nombre, salario FROM empleados;
```

Muestra solo las columnas `nombre` y `salario`.

Ejemplo 3: Usar alias con `AS`

sql
*SELECT nombre AS NombreEmpleado, salario * 12 AS SalarioAnual FROM empleados;*

Cambia el nombre de las columnas en el resultado y calcula un salario anual.

Sentencia `WHERE`

Sintaxis Básica

sql
SELECT [columnas] FROM [nombre_tabla] WHERE [condición];

[condición]: Expresión lógica que filtra los registros.

Operadores Comunes

Operador	**Descripción**	**Ejemplo**
=	Igual	edad = 30
>	Mayor que	salario > 5000
<	Menor que	fecha_nacimiento < '2000-01-01
>=, <=	Mayor o menor o igual	edad >= 18
<> o !=	Distinto	departamento <> 'Ventas'
AND	Y lógico	edad > 25 AND salario < 5000
OR	O lógico	departamento = 'Ventas' OR departamento = 'Marketing
NOT	Negación	NOT (salario > 10000)
IN	Valores específicos	id_empleado IN (101, 102, 103)
BETWEEN	Rango	salario BETWEEN 3000 AND 7000
LIKE	Búsqueda parcial	nombre LIKE 'A%'` (nombres que empiezan con "A")
IS NULL	Valores nulos	email IS NULL

Ejemplos

Ejemplo 1: Filtro simple

sql
SELECT nombre, salario FROM empleados WHERE salario > 3000;

Muestra empleados con salario mayor a 3000.

Ejemplo 2: Múltiples condiciones con `AND`

sql
*SELECT * FROM empleados WHERE departamento = 'Ventas' AND salario < 4000;*

Muestra empleados del departamento de Ventas con salario menor a 4000.

Ejemplo 3: Uso de `IN` y `BETWEEN`

Sql
SELECT nombre FROM empleados WHERE empleado_id IN (1, 2, 3);

SELECT nombre FROM empleados WHERE salario BETWEEN 3000 AND 4500;

SELECT nombre, salario FROM empleados WHERE salario BETWEEN 3000 AND 4500;

Ejemplo 4: Búsqueda parcial con `LIKE`

Sql
SELECT nombre FROM empleados WHERE nombre LIKE 'J%';
```
Muestra nombres que comienzan con "J" (ej. "Juan", "Julia").

### *Ejemplo 5: Filtrar valores nulos*

*Sql*
*SELECT nombre FROM empleados WHERE email IS NULL;*
```
Muestra empleados sin correo electrónico registrado.

Comodines o Wildcards en MySQL

Los comodines (wildcards) se utilizan principalmente en combinación con el operador `LIKE` para realizar búsquedas flexibles dentro de cadenas de texto, especialmente cuando no conocemos completamente el valor que buscamos.

Comodines más usados en MySQL

Comodín	**Descripción**	**Ejemplo de Uso en MySQL**
%	Representa cero, uno o múltiples caracteres	A%' → Coincide con "Ana", "Andrés", "Alejandro"
_	Representa un solo carácter	'A_A' → Coincide con "Ara", "Ama", pero no con "Ana María"
`[char_list]`	No es compatible directamente en MySQL (usado en SQL Server). En MySQL se usa `REGEXP` o `RLIKE`	'A[bc]a' → ‘Ana' REGEXP 'A[bc]a'
`[^char_list]`	No es compatible directamente en MySQL**. Se puede usar `NOT LIKE` o `REGEXP` negado.	'Ana' NOT LIKE 'A%z'

Sintaxis básica con `LIKE`

sql
*SELECT * FROM tabla*
WHERE columna LIKE 'patrón';

Ejemplos prácticos usando comodines

Tabla: `empleados`

id	**nombre**	**apellido**
1	Ana	López
2	Carlos	Mendoza
3	María	Gómez
4	Andrés	Ramírez
5	Lucía	Torres

Usando `%` – Coincide con cualquier cadena

Sql
*SELECT * FROM empleados*
WHERE nombre LIKE 'A%';

→ Devuelve: Ana, Andrés

Usando `%` al inicio – Búsqueda final

Sql
*SELECT * FROM empleados*
WHERE apellido LIKE '%ez';

→ Devuelve apellidos que terminan en "ez", por ejemplo: "Gómez", “Hernández”.

Usando `_` – Un solo carácter

Sql
*SELECT * FROM empleados*
WHERE nombre LIKE '_a_a';

→ Busca nombres de 4 letras donde la segunda y cuarta letra sean "a":
Por ejemplo: "Mara", "Sana", etc.

Usando `NOT LIKE` – Excluir patrones

sql
*SELECT * FROM empleados*
WHERE apellido NOT LIKE 'L%';
```
→ Devuelve todos los empleados cuyo apellido **no empieza por L**

**Usando `REGEXP` – Expresiones regulares avanzadas**

*sql*
*SELECT * FROM empleados*
*WHERE nombre REGEXP '^A';*

→ Nombres que empiezan con A (similar a `LIKE 'A%'`)

*sql*
*SELECT * FROM empleados*
*WHERE nombre REGEXP 'a$';*
```
→ Nombres que terminan con "a"

S*ql*
*SELECT * FROM empleados*

WHERE nombre REGEXP 'A|Lucía';
```

→ Nombres que contienen "A" o son "Lucía"

| Pattern | What the Pattern matches when use REGEXP |
|---|---|
| * | Zero or more instances of string preceding it |
| + | One or more instances of strings preceding it |
| . | Any single character |
| ? | Match zero or one instances of the strings preceding it. |
| ^ | caret(^) matches Beginning of string |
| $ | End of string |
| [abc] | Any character listed between the square brackets |
| [^abc] | Any character not listed between the square brackets |
| [A-Z] | match any upper case letter. |
| [a-z] | match any lower case letter |
| [0-9] | match any digit from 0 through to 9. |
| [[:<:]] | matches the beginning of words. |
| [[:>:]] | matches the end of words. |
| [:class:] | matches a character class i.e. [:alpha:] to match letters, [:space:] to match white space, [:punct:] is match punctuations and [:upper:] for upper class letters. |
| p1\|p2\|p3 | Alternation; matches any of the patterns p1, p2, or p3 |
| {n} | Exactly n instances of preceding element |
| {m,n} | between m and n instances of preceding element |

**Notas importantes:**

- El operador `LIKE` es **case-insensitive** (no distingue mayúsculas/minúsculas)
- Para búsquedas complejas, `REGEXP` ofrece mucha más potencia que `LIKE`.
```

Ejercicios Prácticos

Escenario: Tabla empleados

id_empleado	nombre	departamento	salario	fecha_nacimiento
101	Ana	Marketing	4500	1990-05-15
102	Luis	Ventas	3800	1985-08-22
103	Carlos	IT	6000	1995-03-10
104	María	Ventas	5200	1980-11-30
105	Pedro	IT	7000	1992-01-01

Preguntas

1. Mostrar todos los empleados del departamento de IT.
2. Listar nombres de empleados con salario entre 4000 y 6000.
3. Encontrar empleados nacidos después del año 1990.
4. Mostrar empleados cuyo nombre termine con "s" (ej. Luis).

Respuestas

sql
1.
*SELECT * FROM empleados WHERE departamento = 'IT';*

2.
SELECT nombre FROM empleados WHERE salario BETWEEN 4000 AND 6000;

3.
SELECT nombre FROM empleados WHERE fecha_nacimiento > '1990-12-31';

4.
SELECT nombre FROM empleados WHERE nombre LIKE '%s';

Buenas Prácticas

- Especificar columnas: En lugar de `SELECT *`, usar `SELECT columna1, columna2` para mejorar rendimiento.
- Alias claros: Usar `AS` para dar nombres significativos a columnas calculadas.
- Comentarios: Documentar consultas complejas con `--`.
- Optimización: Evitar condiciones innecesarias en `WHERE` para reducir carga.

Referencias

- Documentación oficial de MySQL: https://dev.mysql.com/doc/
- Libro: *SQL para Principiantes* de John Doe (Ejemplo hipotético).