# SQL

## *Data Manipulation Language (DML)*

CertiDevs

# Índice de contenidos

# 1. Introducción a sentencias DML

DML son las siglas de Data Manipulation Language, y conforman todas aquellas sentencias disponibles para recuperar datos de una base de datos, así como insertar, modificar y eliminar datos.

- SELECT: proyección y selección
- INSERT: inserción de datos
- UPDATE: modificación de datos
- DELETE: eliminación de datos

# 2. SELECT: Proyección y selección

El comando SELECT en SQL es utilizado para consultar datos de una o varias tablas. La proyección se refiere a la selección de columnas específicas, mientras que la selección se refiere a la elección de filas que cumplan ciertas condiciones.

A continuación, se detallan cómo utilizar el comando SELECT para la proyección y la selección en SQL con ejemplos.

## 2.1. Proyección

La proyección se realiza especificando las columnas que se desean seleccionar en la consulta. La sintaxis básica es la siguiente:

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

Por ejemplo, para seleccionar las columnas first_name y last_name de la tabla employees:

```
SELECT first_name, last_name
FROM employees;
```

Si deseas seleccionar todas las columnas de la tabla, utiliza el asterisco (*) en lugar de enumerar las columnas:

```
SELECT * FROM employees;
```

## 2.2. Cláusulas y operadores de SQL

- WHERE: filtrado de datos
- AND, OR, NOT: operadores lógicos

- LIKE, IN, BETWEEN: operadores de comparación
- ORDER BY: ordenamiento de resultados
- GROUP BY: agrupación de resultados
- HAVING: filtrado de resultados agrupados
- LIMIT, OFFSET: paginación de resultados

### 2.2.1. Cláusula WHERE

La selección de filas se realiza utilizando la cláusula WHERE, que permite especificar condiciones que deben cumplir las filas para ser incluidas en el resultado. La sintaxis básica es la siguiente:

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

Por ejemplo, para seleccionar las columnas first_name, last_name y salary de la tabla employees donde el salario es mayor que 50000.

```
SELECT first_name, last_name, salary
FROM employees
WHERE salary > 50000;
```

### 2.2.2. Operadores WHERE

Se pueden usar operadores de comparación en la cláusula WHERE, como `=`, `<>` o `!=`, `<`, `>`, `<=`, y `>=`. Además, se pueden combinar múltiples condiciones utilizando los operadores lógicos `AND`, `OR` y `NOT`.

Por ejemplo, para seleccionar las columnas first_name, last_name y salary de la tabla employees donde el salario está entre 50,000 y 100,000:

```
SELECT first_name, last_name, salary
FROM employees
WHERE salary >= 50000 AND salary <= 100000;
```

También puedes utilizar la cláusula WHERE para realizar comparaciones de texto. Por ejemplo, para seleccionar las columnas first_name, last_name y job_title de la tabla employees donde el título del trabajo contiene la palabra "Manager":

```
SELECT first_name, last_name, job_title
FROM employees
WHERE job_title LIKE '%Manager%';
```

### 2.2.3. Operadores OR y NOT

Los operadores OR y NOT pueden usarse en la cláusula WHERE para combinar condiciones de diferentes maneras. El operador OR devuelve verdadero si alguna de las condiciones es verdadera, mientras que el operador NOT invierte el resultado de una condición.

Por ejemplo, para seleccionar las columnas first_name, last_name y job_title de la tabla employees donde el título del trabajo es "Manager" o "Assistant Manager":

```
SELECT first_name, last_name, job_title
FROM employees
WHERE job_title = 'Manager' OR job_title = 'Assistant Manager';
```

Para seleccionar las columnas first_name, last_name y job_title de la tabla employees donde el título del trabajo NO es "Manager":

```
SELECT first_name, last_name, job_title
FROM employees
WHERE NOT job_title = 'Manager';
```

### 2.2.4. Operador IN

El operador IN permite especificar múltiples valores en la cláusula WHERE y es una forma abreviada de usar múltiples condiciones OR.

Por ejemplo, para seleccionar las columnas first_name, last_name y job_title de la tabla employees donde el título del trabajo es "Manager" o "Assistant Manager" o "Salesperson":

```
SELECT first_name, last_name, job_title
FROM employees
WHERE job_title IN ('Manager', 'Assistant Manager', 'Salesperson');
```

### 2.2.5. Operador BETWEEN

El operador BETWEEN se utiliza para seleccionar valores dentro de un rango. La sintaxis es:

```
SELECT column1, column2, ...
FROM table_name
WHERE column BETWEEN value1 AND value2;
```

Por ejemplo, para seleccionar las columnas first_name, last_name y salary de la tabla employees donde el salario está entre 50,000 y 100,000:

```
SELECT first_name, last_name, salary
FROM employees
```

```
WHERE salary BETWEEN 50000 AND 100000;
```

### 2.2.6. Operador LIKE

El operador LIKE se utiliza en la cláusula WHERE para buscar un patrón específico en una columna. Los caracteres comodín más comunes son el porcentaje (%) que representa cero, uno o varios caracteres, y el guion bajo (_) que representa un solo carácter. La sintaxis básica es:

```
SELECT column1, column2, ...
FROM table_name
WHERE column LIKE pattern;
```

Por ejemplo, para seleccionar las columnas first_name, last_name y job_title de la tabla employees donde el título del trabajo comienza con "Manager":

```
SELECT first_name, last_name, job_title
FROM employees
WHERE job_title LIKE 'Manager%';
```

Para seleccionar las columnas first_name, last_name y email de la tabla employees donde el correo electrónico contiene exactamente tres caracteres después del símbolo '@':

```
SELECT first_name, last_name, email
FROM employees
WHERE email LIKE '%@___%';
```

### 2.2.7. Cláusula ORDER BY

La cláusula ORDER BY se utiliza para ordenar los resultados de una consulta. Puedes ordenar los resultados en orden ascendente (ASC) o descendente (DESC). La sintaxis básica es:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column_name ASC|DESC;
```

Por ejemplo, para seleccionar las columnas first_name, last_name y salary de la tabla employees y ordenar los resultados por salario en orden descendente:

```
SELECT first_name, last_name, salary
FROM employees
ORDER BY salary DESC;
```

### 2.2.8. Cláusula LIMIT

La cláusula LIMIT se utiliza para limitar la cantidad de resultados que devuelve una consulta. La sintaxis básica es:

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

Por ejemplo, para seleccionar las columnas first_name, last_name y salary de la tabla employees y limitar los resultados a 5 filas:

```
SELECT first_name, last_name, salary
FROM employees
LIMIT 5;
```

### 2.2.9. Cláusula OFFSET

La cláusula OFFSET se utiliza junto con la cláusula LIMIT para especificar el punto de inicio para la consulta. La sintaxis básica es:

```
SELECT column1, column2, ...
FROM table_name
LIMIT number OFFSET start;
```

Por ejemplo, para seleccionar las columnas first_name, last_name y salary de la tabla employees, limitar los resultados a 5 filas y comenzar desde la fila 11:

```
SELECT first_name, last_name, salary
FROM employees
LIMIT 5 OFFSET 10;
```

# 3. INSERT: Inserción de datos

La sentencia INSERT se utiliza para insertar nuevos registros en una tabla. Hay dos formas principales de utilizar la sentencia INSERT: insertar un solo registro o insertar múltiples registros a la vez.

## 3.1. Inserción de un único registro

Para insertar un único registro en una tabla, se utiliza la siguiente sintaxis:

```
INSERT INTO table_name (column1, column2, column3, ...)
```

```
VALUES (value1, value2, value3, ...);
```

Ejemplo: Insertar un nuevo empleado en la tabla employees:

```
INSERT INTO employees (first_name, last_name, birthdate, department_id)
VALUES ('John', 'Doe', '1985-03-20', 2);
```

## 3.2. Inserción de múltiples registros

Para insertar varios registros a la vez en una tabla, se puede utilizar la siguiente sintaxis:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES
(value1_1, value1_2, value1_3, ...),
(value2_1, value2_2, value2_3, ...),
...;
```

Ejemplo: Insertar varios empleados en la tabla employees:

```
INSERT INTO employees (first_name, last_name, birthdate, department_id)
VALUES
('Jane', 'Doe', '1980-05-15', 1),
('Alice', 'Johnson', '1990-08-30', 3),
('Bob', 'Smith', '1978-12-10', 2);
```

La sentencia INSERT es esencial para agregar datos a las tablas en una base de datos SQL. Se pueden insertar registros individuales o múltiples registros a la vez, lo que permite una gran flexibilidad y eficiencia en la gestión de datos.

# 4. UPDATE: Modificación de datos

La sentencia UPDATE se utiliza para modificar datos existentes en una tabla. Puedes actualizar uno o varios registros a la vez, y también puedes utilizar condiciones para seleccionar qué registros deben actualizarse.

## 4.1. Actualización de un único registro

Para actualizar un único registro en una tabla, se utiliza la siguiente sintaxis:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Ejemplo: Actualizar el apellido y el departamento de un empleado específico en la tabla employees:

```
UPDATE employees
SET last_name = 'Doe-Smith', department_id = 4
WHERE employee_id = 1;
```

## 4.2. Actualización de múltiples registros

Para actualizar varios registros a la vez en una tabla, puedes utilizar la misma sintaxis que para un único registro, pero con una condición que coincida con varios registros.

Ejemplo: Aumentar el salario de todos los empleados en un 10% en la tabla employees:

```
UPDATE employees
SET salary = salary * 1.1;
```

Ten en cuenta que en este ejemplo no se incluye una cláusula WHERE, lo que significa que la actualización se aplicará a todos los registros de la tabla.

Si deseas actualizar solo registros específicos, puedes agregar una cláusula WHERE con una condición:

Ejemplo: Aumentar el salario de todos los empleados del departamento 2 en un 15% en la tabla employees:

```
UPDATE employees
SET salary = salary * 1.15
WHERE department_id = 2;
```

La sentencia UPDATE es fundamental para modificar los datos existentes en las tablas de una base de datos SQL. Puedes actualizar uno o varios registros a la vez y utilizar condiciones para determinar qué registros deben actualizarse.

# 5. DELETE: Eliminación de datos en SQL

La sentencia DELETE se utiliza para eliminar registros existentes en una tabla. Puedes eliminar uno o varios registros a la vez, y también puedes utilizar condiciones para seleccionar qué registros deben eliminarse.

## 5.1. Eliminación de un único registro

Para eliminar un único registro en una tabla, se utiliza la siguiente sintaxis:

```
DELETE FROM table_name
```

```
WHERE condition;
```

Ejemplo: Eliminar un empleado específico de la tabla employees:

```
DELETE FROM employees
WHERE employee_id = 1;
```

## 5.2. Eliminación de múltiples registros

Para eliminar varios registros a la vez en una tabla, puedes utilizar la misma sintaxis que para un único registro, pero con una condición que coincida con varios registros:

Ejemplo: Eliminar todos los empleados del departamento 3 en la tabla employees:

```
DELETE FROM employees
WHERE department_id = 3;
```

Ten en cuenta que si no incluyes una cláusula WHERE, la sentencia DELETE eliminará todos los registros de la tabla. Por lo general, esto no es lo que se desea, así que asegúrate de incluir siempre una cláusula WHERE cuando utilices DELETE.

Ejemplo de eliminación de todos los registros de una tabla (no recomendado en la mayoría de los casos):

```
DELETE FROM employees;
```

La sentencia DELETE es fundamental para eliminar datos existentes en las tablas de una base de datos SQL. Puedes eliminar uno o varios registros a la vez y utilizar condiciones para determinar qué registros deben eliminarse.

# 6. Consultas JOIN

Cuando se trabaja con múltiples tablas relacionadas a través de claves primarias y foráneas, es común necesitar consultar datos de múltiples tablas a la vez.

Es por tanto que se utilizan consultas cruzadas, haciendo uso de consultas de tipo JOIN.

## 6.1. INNER JOIN

INNER JOIN es una operación clave en SQL que permite combinar registros de dos o más tablas basándose en una columna común entre ellas. El resultado de un INNER JOIN incluye solo aquellos registros que tienen coincidencias en ambas tablas. Si no hay coincidencias, el registro no aparecerá en el conjunto de resultados.

A continuación, se explica en detalle el uso de INNER JOIN en SQL.

Supongamos que tenemos las siguientes dos tablas:

- orders: contiene información sobre pedidos realizados por clientes.
- customers: contiene información sobre los clientes que realizan pedidos.

```
CREATE TABLE customers (
customer_id INT PRIMARY KEY,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL,
email VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE orders (
order_id INT PRIMARY KEY,
order_date DATE NOT NULL,
customer_id INT NOT NULL,
FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Para combinar registros de las tablas orders y customers en función de la columna customer_id común, podemos usar INNER JOIN de la siguiente manera:

```
SELECT
orders.order_id,
orders.order_date,
customers.first_name,
customers.last_name,
customers.email
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id;
```

La cláusula INNER JOIN va después de la cláusula FROM y antes de la cláusula WHERE (si la hay). La condición de unión se especifica después de la palabra clave ON. En este caso, la condición de unión es orders.customer_id = customers.customer_id. Esto indica que los registros de ambas tablas deben coincidir en la columna customer_id.

El resultado del INNER JOIN incluirá solo aquellos registros de orders y customers que tienen coincidencias en la columna customer_id. Si un registro en una tabla no tiene una coincidencia en la otra tabla, no se incluirá en el conjunto de resultados.

En resumen, INNER JOIN en SQL es una operación esencial para combinar registros de múltiples tablas basándose en una columna común. El resultado de un INNER JOIN incluye solo registros que tienen coincidencias en ambas tablas, lo que permite obtener información más completa y relacionada de diferentes tablas en un solo conjunto de resultados.

## 6.2. LEFT JOIN

Además del INNER JOIN, SQL proporciona otras operaciones de unión para combinar registros de dos o más tablas incluso si no hay coincidencias entre ellos. Estas operaciones son LEFT JOIN, RIGHT JOIN y FULL OUTER JOIN.

A continuación, se explica en detalle cada tipo de unión:

LEFT JOIN: devuelve todos los registros de la tabla izquierda y las coincidencias de la tabla derecha. Si no hay coincidencias en la tabla derecha, se mostrarán valores NULL.

```
SELECT
customers.customer_id,
customers.first_name,
customers.last_name,
orders.order_id,
orders.order_date
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

En este ejemplo, la consulta devuelve todos los registros de la tabla customers y las coincidencias de la tabla orders. Si un cliente no tiene pedidos, se mostrarán valores NULL en las columnas de orders.

## 6.3. RIGHT JOIN

RIGHT JOIN: devuelve todos los registros de la tabla derecha y las coincidencias de la tabla izquierda. Si no hay coincidencias en la tabla izquierda, se mostrarán valores NULL.

```
SELECT
customers.customer_id,
customers.first_name,
customers.last_name,
orders.order_id,
orders.order_date
FROM customers
RIGHT JOIN orders ON customers.customer_id = orders.customer_id;
```

En este ejemplo, la consulta devuelve todos los registros de la tabla orders y las coincidencias de la tabla customers. Si un pedido no tiene un cliente asociado, se mostrarán valores NULL en las columnas de customers.

## 6.4. FULL OUTER JOIN

FULL OUTER JOIN: devuelve todos los registros de ambas tablas, con coincidencias donde sea posible y valores NULL en las columnas donde no haya coincidencias. (Tenga en cuenta que FULL

OUTER JOIN no está soportado en MySQL).

```
SELECT
customers.customer_id,
customers.first_name,
customers.last_name,
orders.order_id,
orders.order_date
FROM customers
FULL OUTER JOIN orders ON customers.customer_id = orders.customer_id;
```

En este ejemplo, la consulta devuelve todos los registros de ambas tablas, customers y orders. Si un cliente no tiene pedidos, se mostrarán valores NULL en las columnas de orders. Si un pedido no tiene un cliente asociado, se mostrarán valores NULL en las columnas de customers.

# 7. Subconsultas con SELECT

Las subconsultas son consultas anidadas dentro de otras consultas. Pueden ser utilizadas en diferentes partes de una consulta SQL como en la cláusula SELECT, FROM, WHERE o HAVING. Las subconsultas pueden ayudar a dividir consultas complejas en partes más simples y manejables, lo que facilita la resolución de problemas y la optimización del rendimiento.

A continuación, se explica cómo usar subconsultas con SELECT en diferentes partes de la consulta principal.

## 7.1. Subconsultas en la cláusula SELECT:

Las subconsultas en la cláusula SELECT se utilizan para calcular valores dinámicos basados en otros valores de la misma fila o de otras tablas. Estas subconsultas deben devolver un único valor.

```
SELECT
customers.customer_id,
customers.first_name,
customers.last_name,
(SELECT COUNT(orders.order_id)
FROM orders
WHERE orders.customer_id = customers.customer_id) AS order_count
FROM customers;
```

En este ejemplo, la subconsulta calcula el número de pedidos para cada cliente y se muestra en la columna order_count.

## 7.2. Subconsultas en la cláusula FROM:

Las subconsultas en la cláusula FROM se utilizan para crear una tabla temporal que puede ser utilizada en la consulta principal.

```
SELECT
order_summary.customer_id,
customers.first_name,
customers.last_name,
order_summary.order_count
FROM
(SELECT customer_id, COUNT(order_id) AS order_count
FROM orders
GROUP BY customer_id) AS order_summary
JOIN customers ON order_summary.customer_id = customers.customer_id;
```

En este ejemplo, la subconsulta crea una tabla temporal llamada order_summary que contiene el número de pedidos por cliente. Luego, esta tabla temporal se une con la tabla customers para mostrar el nombre y apellido de cada cliente junto con su número de pedidos.

## 7.3. Subconsultas en la cláusula WHERE o HAVING:

Las subconsultas en la cláusula WHERE o HAVING se utilizan para filtrar filas en función de los resultados de otras consultas.

```
SELECT
customers.customer_id,
customers.first_name,
customers.last_name
FROM customers
WHERE customers.customer_id IN
(SELECT DISTINCT customer_id
FROM orders
WHERE order_date >= '2021-01-01');
```

En este ejemplo, la subconsulta devuelve una lista de customer_id que tienen pedidos desde '2021-01-01'. La consulta principal muestra los clientes cuyos ID están en esa lista.