SQL

SQL, o Structured Query Language, es un lenguaje de programación especializado utilizado para administrar y manipular bases de datos relacionales. Fue desarrollado inicialmente por IBM en la década de 1970 y ha sido ampliamente adoptado como estándar de facto para interactuar con bases de datos. SQL permite realizar una variedad de operaciones en bases de datos, como consultas para recuperar datos, inserción de nuevos datos, actualización de datos existentes y eliminación de datos, entre otras funciones. La norma ISO/IEC 9075 especifica el estándar de SQL, que ha pasado por varias revisiones a lo largo de los años.

Índice

Ejercicios Interesantes

- Apuntes SQL

Operaciones en SQL

- **SELECT:** Es utilizado para recuperar datos de una base de datos. Permite seleccionar uno o más campos de una o más tablas, opcionalmente aplicando condiciones de filtrado y ordenamiento.
- **INSERT:** Es utilizado para agregar nuevos registros a una tabla. Permite especificar los valores que se desean insertar en las columnas de la tabla.

- **UPDATE:** Es utilizado para modificar los registros existentes en una tabla. Permite cambiar los valores de una o más columnas en función de una condición de filtro.
- **DELETE:** Es utilizado para eliminar registros de una tabla. Permite eliminar uno o más registros que cumplan con ciertas condiciones especificadas.

Estas operaciones son fundamentales para consultar, agregar, modificar y eliminar datos en una base de datos relacional utilizando SQL.

Creacción de Base de datos

- **CREATE TABLE:** Esta sentencia se utiliza para crear una nueva tabla en la base de datos. Al crear una tabla, especificamos el nombre de la tabla y las columnas que contendrá, junto con sus tipos de datos y cualquier restricción que deseemos aplicar.

```
CREATE TABLE Empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    edad INT CHECK (edad > 0),
    rol VARCHAR(20) DEFAULT 'empleado'
);
```

- **ALTER TABLE:** Esta sentencia se utiliza para realizar cambios en una tabla existente, como agregar, modificar o eliminar columnas, así como agregar o quitar restricciones.

```
ALTER TABLE personas ADD APELLIDOS VARCHAR(50);
ALTER TABLE personas RENAME COLUMN APELLIDOS TO APELLIDOS_PERSONA;
ALTER TABLE personas MODIFY COLUMN APELLIDOS_PERSONA VARCHAR(30);
ALTER TABLE personas DROP COLUMN APELLIDOS_PERSONA;
```

- **DROP TABLE:** Esta sentencia se utiliza para eliminar una tabla existente en base de datos.

```
DROP TABLE IF EXISTS personas;
```

Claves e Índices

- **PRIMARY KEY:** Una clave primaria (PRIMARY KEY) en una tabla de base de datos es una columna o un conjunto de columnas que se utiliza para identificar de forma única cada fila de la tabla. La clave primaria garantiza la integridad de los datos y evita la inserción de filas duplicadas. Cada tabla puede tener solo una clave primaria y no puede contener valores nulos.
- **FOREIGN KEY:** Una clave foránea (FOREIGN KEY) en una tabla de base de datos establece una relación entre dos tablas. La clave foránea es una columna o un conjunto de columnas que establece una

relación de referencia entre los datos de una tabla (tabla hija) y los datos de otra tabla (tabla padre). La clave foránea garantiza la integridad referencial y puede utilizarse para realizar acciones como CASCADE, RESTRICT, SET NULL, etc., cuando se realizan operaciones de actualización o eliminación en la tabla padre.

```
CREATE TABLE CURSO (
    CURSO_ID INT PRIMARY KEY,
    NOMBRE_CURSO VARCHAR(50)
);

CREATE TABLE ESTUDIANTE (
    ESTUDIANTE_ID INT PRIMARY KEY,
    NOMBRE VARCHAR(50),
    EDAD INT,
    CURSO_ID INT,
    CONSTRAINT FK_CURSO -- Nombre de la restricción de clave externa
    FOREIGN KEY (CURSO_ID) REFERENCES CURSO(CURSO_ID)
);
```

```
CREATE TABLE CURSO (
    CURSO_ID INT,
    ANIO INT,
    NOMBRE_CURSO VARCHAR(50),
    PRIMARY KEY (CURSO_ID, ANIO)
);

CREATE TABLE ESTUDIANTE (
    ESTUDIANTE_ID INT,
    NOMBRE VARCHAR(50),
    EDAD INT,
    CURSO_ID INT,
    CURSO_ANIO INT,
    PRIMARY KEY (ESTUDIANTE_ID, CURSO_ID), -- Clave primaria compuesta
    CONSTRAINT FK_CURSO -- Nombre de la restricción de clave externa
    FOREIGN KEY (CURSO_ID, CURSO_ANIO) REFERENCES CURSO(CURSO_ID, ANIO) -- Clave
externa compuesta
);
```

- **Índices**: Un índice en SQL es una estructura utilizada para mejorar la eficiencia de las consultas en una base de datos. Básicamente, es una lista ordenada de valores de una o más columnas de una tabla que permite al motor de la base de datos encontrar rápidamente los registros que cumplen con ciertos criterios de búsqueda.

```
CREATE INDEX idx_nombre_indice
ON tabla_ejemplo (columna);
```

```
CREATE INDEX idx_nombre_indice
ON tabla_ejemplo (columna1, columna2);
```

Tipos de datos

Datos numéricos:

- **INT**: Almacena números enteros con un rango de aproximadamente -2 mil millones a +2 mil millones.
- **BIGINT**: Similar a INT pero con un rango mucho más grande, útil para números muy grandes.
- **SMALLINT**: Almacena números enteros en un rango más pequeño que INT.
- **TINYINT**: Almacena números enteros en un rango aún más pequeño que SMALLINT.
- **BIT**: Almacena valores de tipo booleano, es decir, 0 o 1.
- **DECIMAL / NUMERIC**: Almacena números con una precisión fija y escala, útil para valores monetarios y cálculos financieros.
- **MONEY**: Similar a DECIMAL pero optimizado para valores monetarios.
- **SMALLMONEY**: Una versión más pequeña de MONEY.
- **FLOAT / REAL**: Almacenan números de punto flotante, útiles para valores que requieren precisión decimal variable.

Cadenas de caracteres:

- **CHAR**: Almacena cadenas de caracteres de longitud fija.
- **VARCHAR**: Almacena cadenas de caracteres de longitud variable.
- **TEXT**: Almacena datos de texto de longitud variable (será eliminado en futuras versiones de SQL Server).

Cadenas de caracteres Unicode:

- **NCHAR**: Similar a CHAR pero para caracteres Unicode.
- **NVARCHAR**: Similar a VARCHAR pero para caracteres Unicode.
- **NTEXT**: Similar a TEXT pero para caracteres Unicode (será eliminado en futuras versiones de SQL Server).

Cadenas binarias:

- **BINARY**: Almacena datos binarios de longitud fija.
- **VARBINARY**: Almacena datos binarios de longitud variable.
- **IMAGE**: Almacena datos binarios de longitud variable (será eliminado en futuras versiones de SQL Server).

Fecha y hora:

- **DATE**: Almacena fechas sin incluir la hora.
- **DATETIME**: Almacena fechas y horas.
- **SMALLDATETIME**: Almacena fechas y horas pero con menor precisión que DATETIME.
- **DATETIME2**: Almacena fechas y horas con mayor precisión que DATETIME.
- **DATETIMEOFFSET**: Almacena fechas y horas con información sobre la zona horaria.
- **TIME**: Almacena horas sin incluir la fecha.

Restricciones

- **NOT NULL:** Esta restricción asegura que los valores en la columna no pueden ser nulos (es decir, deben tener algún valor). Por ejemplo, si definimos una columna "edad" con la restricción NOT NULL, cada fila de la tabla debe tener un valor de edad y no puede estar vacía.
- **UNIQUE:** Esta restricción asegura que cada valor en la columna sea único en toda la tabla. Es decir, no puede haber dos filas con el mismo valor en esa columna. Por ejemplo, si definimos una columna "email" con la restricción UNIQUE, no puede haber dos personas con el mismo correo electrónico en la tabla.
- **CHECK:** Esta restricción permite definir una condición que debe cumplir el valor de la columna. Si la condición no se cumple, la inserción o actualización de la fila se rechazará. Por ejemplo, si queremos asegurarnos de que la columna "edad" solo contenga valores positivos, podemos usar una restricción CHECK.
- **DEFAULT:** Esta restricción permite especificar un valor predeterminado para la columna cuando no se proporciona ningún valor durante la inserción. Por ejemplo, si queremos que la columna "rol" tenga un valor predeterminado de "empleado" cuando se inserta una nueva fila sin especificar el rol, podemos usar una restricción DEFAULT.

```
CREATE TABLE Empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    edad INT CHECK (edad > 0),
    rol VARCHAR(20) DEFAULT 'empleado'
);
```

IS / IS NULL

- `IS / IS NULL` Esta sentencia se utiliza principalmente en combinación con NULL para verificar si un valor es nulo o no.

```
SALARIO IS NULL
```

```
SALARIO IS NOT NULL
```

- La siguiente consulta selecciona todos los datos de los productos cuyo precio no sea nulo.

```
SELECT *
FROM VENTAS.PRODUCTOS
WHERE PRECIO_ACTUAL IS NOT NULL;
```

IFNULL(COMISION, 0)

- `IFNULL(COMISION, 0)` Esta sentencia se utiliza para proporcionar un valor predeterminado (en este caso, 0) cuando el valor de COMISION es nulo.

```
IFNULL(COMISION, 0)
```

- La siguiente consulta selecciona el apellido, salario y comisión de los empleados de la tabla EMPLEADOS en la base de datos VENTAS, y calcula una columna adicional llamada SALARIO_COMISION, que es la suma del salario y la comisión, donde la comisión se maneja de forma segura en caso de ser nula.

```
SELECT
    APELLIDO,
    SALARIO,
    COMISION,
    (SALARIO + IFNULL(COMISION, 0)) AS SALARIO_COMISION
FROM VENTAS.EMPLEADOS;
```

IN(X,Y,Z)

- `IN(X,Y,Z)` Esta sentencia se utiliza para comparar un valor con una lista de posibles valores.

```
LOCALIDAD IN ('MADRID', 'BARCELONA', 'VALENCIA');
```

- La siguiente consulta selecciona el apellido, la localidad, el oficio y el salario de los empleados de la tabla EMPLEADOS y la localidad de los departamentos de la tabla DEPARTAMENTOS en la base de datos VENTAS, donde el número de departamento (DEP_NO) coincide entre ambas tablas y la localidad del departamento está entre Madrid, Barcelona o Valencia.

```
SELECT E.APELLIDO, D.LOCALIDAD, E.OFICIO, E.SALARIO
FROM VENTAS.EMPLEADOS E, VENTAS.DEPARTAMENTOS D
WHERE
    E.DEP_NO = D.DEP_NO AND
    D.LOCALIDAD IN ('MADRID', 'BARCELONA', 'VALENCIA');
```

CONCAT(X,Y)

- `CONCAT(X,Y)` Esta sentencia se utiliza para concatenar dos o más cadenas de texto en una sola cadena.

```
CONCAT('Nombre: ', NOMBRE):
```

- La siguiente consulta combina el nombre y la localidad de los clientes en una sola cadena, creando una nueva columna llamada 'INFORMACION DEL CLIENTE'.

```
SELECT CONCAT(
        'Nombre: ', NOMBRE, ' y es de: ', LOCALIDAD
    ) AS `INFORMACION DEL CLIENTE`
FROM CLIENTES;
```

BETWEEN

- `BETWEEN` Esta sentencia permite filtrar resultados dentro de un rango específico. Se utiliza para comparar si un valor se encuentra dentro de un rango dado, inclusive de los límites del rango.

```
SALARIO BETWEEN 1500 AND 2500;
```

- La siguiente consulta selecciona todos los empleados cuyo salario está dentro del rango de 1500 y 2500, incluyendo ambos límites del rango.

```
SELECT * FROM EMPLEADOS WHERE SALARIO BETWEEN 1500 AND 2500;
```

LEFT(X,pos)

- `LEFT(X,pos)` Esta sentencia devuelve un número específico de caracteres desde el comienzo de una cadena de texto.

```
LEFT(APELLIDO, 1) = 'M'
```

- La siguiente consulta selecciona todos los empleados de la tabla EMPLEADOS en el esquema VENTAS cuyo apellido comienza con la letra 'M'.

```
SELECT *
FROM VENTAS.EMPLEADOS
WHERE
    LEFT(APELLIDO, 1) = 'M';
```

RIGHT(X,pos)

- `RIGHT(X,pos)` Esta sentencia devuelve devuelve un número específico de caracteres desde el final de una cadena de texto.

```
RIGHT(APELLIDO, 1) = 'M'
```

- La siguiente consulta selecciona todos los empleados de la tabla EMPLEADOS en el esquema VENTAS cuyo apellido termina con la letra 'M'.

```
SELECT *
FROM VENTAS.EMPLEADOS
WHERE
    RIGHT(APELLIDO, 1) = 'M';
```

DISTINCT

- `DISTINCT` Esta sentencia se utiliza para seleccionar valores únicos de una o más columnas en un conjunto de resultados.

```
DISTINCT VENDEDOR_NO, LOCALIDAD
```

- La siguiente consulta selecciona valores únicos de las columnas VENDEDOR_NO y LOCALIDAD de la tabla CLIENTES.

```
SELECT DISTINCT VENDEDOR_NO, LOCALIDAD FROM CLIENTES;
```

ALL

- `ALL` Esta sentencia se utiliza en combinación con operadores de comparación, como =, >, <, etc. Sirve para comparar un valor con todos los valores de un conjunto de resultados devuelto por una subconsulta. La subconsulta puede devolver múltiples valores, y la comparación se evaluará como verdadera si la condición especificada es verdadera para todos los valores devueltos por la subconsulta.

- La siguiente consulta selecciona los empleados cuyos salarios totales son mayores que el salario total de cualquier empleado en el departamento número 30.

```
SELECT *
FROM VENTAS.EMPLEADOS
WHERE (SALARIO + IFNULL(COMISION, 0)) > ALL (
        SELECT SALARIO + IFNULL(COMISION, 0)
```

```
        FROM VENTAS.EMPLEADOS
        WHERE
            DEP_NO = 30
    );
```

CASE

- `CASE` Esta sentencia permite realizar evaluaciones condicionales dentro de una consulta y devolver un valor específico basado en esas condiciones.
- La siguiente consulta selecciona el nombre y la puntuación de los estudiantes de la tabla ESTUDIANTES, y utiliza la expresión CASE para asignar una nota ('A', 'B', 'C' o 'D') a cada estudiante en función de su puntuación. La columna resultante se etiqueta como 'NOTA'.

```
SELECT
    NOMBRE,
    PUNTUACION,
    CASE
        WHEN PUNTUACION >= 90 THEN 'A'
        WHEN PUNTUACION >= 80 THEN 'B'
        WHEN PUNTUACION >= 70 THEN 'C'
        ELSE 'D'
    END AS 'NOTA'
FROM ESTUDIANTES;
```

ORDER BY

- `ORDER BY` Esta sentencia ordenará los resultados por uno o varios campos en orden ascendente o descendente.

```
ORDER BY COMISION ASC
```

```
ORDER BY COMISION DESC
```

```
ORDER BY Edad ASC, Salario DESC
```

- La siguiente consulta selecciona el nombre, edad y salario de los empleados de la tabla "Empleados", y ordena los resultados primero por edad en orden ascendente (ASC), y luego por salario en orden descendente (DESC).

```
SELECT Nombre, Edad, Salario
FROM Empleados
ORDER BY Edad ASC, Salario DESC;
```

GROUP BY

- `GROUP BY` Esta sentencia se utiliza para agrupar filas que comparten un valor común en una o más columnas, permitiendo luego aplicar funciones de agregación como SUM(), COUNT(), AVG(), entre otras, sobre los grupos resultantes.

```
GROUP BY X,Y,Z;
```

```
HAVING X > 2;
```

- La siguiente consulta selecciona el nombre de cada departamento y la media salarial de ese departamento, para los departamentos cuya media salarial es mayor que el salario del empleado 7654.

```
SELECT D.DNOMBRE, AVG(SALARIO) AS 'MEDIA SALARIAL'
FROM VENTAS.DEPARTAMENTOS D
    INNER JOIN VENTAS.EMPLEADOS E ON D.DEP_NO = E.DEP_NO
GROUP BY
    D.DEP_NO
HAVING
    AVG(SALARIO) > (
        SELECT SALARIO
        FROM VENTAS.EMPLEADOS
        WHERE
            EMP_NO = '7654'
    );
```

Tipos de Joins

- **INNER JOIN:** Devuelve solo los registros que tienen coincidencias en ambas tablas. Es decir, solo devuelve filas cuando la condición de unión se cumple en ambas tablas.
- **LEFT JOIN:** Devuelve todos los registros de la tabla izquierda (primera tabla especificada después de FROM), junto con los registros coincidentes de la tabla derecha (segunda tabla especificada después de JOIN). Si no hay coincidencias en la tabla derecha, se devuelven valores NULL.
- **RIGHT JOIN:** Devuelve todos los registros de la tabla derecha, junto con los registros coincidentes de la tabla izquierda. Si no hay coincidencias en la tabla izquierda, se devuelven valores NULL.

```
SELECT EMP_NO, APELLIDO, DEP_NO FROM VENTAS.EMPLEADOS;
```

Resultado

EMP_NO	APELLIDO	DEP_NO
1	REY	10
6	CALVO	30
38	GIL	20
7499	ALONSO	30
7521	LOPEZ	10
7654	MARTIN	30
7698	GARRIDO	30
7782	MARTINEZ	10
7839	REY	10
7844	CALVO	30
7876	GIL	20
7900	JIMENEZ	20
7902	GARRIDO	NULL
8998	CORTES	30

INNER JOIN

```
SELECT EMP_NO, APELLIDO, D.DEP_NO, DNOMBRE
FROM VENTAS.EMPLEADOS E
    INNER JOIN VENTAS.DEPARTAMENTOS D ON E.DEP_NO = D.DEP_NO;
```

EMP_NO	APELLIDO	DEP_NO	DEPARTAMENTO
1	REY	10	CONTABILIDAD
7521	LOPEZ	10	CONTABILIDAD
7782	MARTINEZ	10	CONTABILIDAD
7839	REY	10	CONTABILIDAD
38	GIL	20	INVESTIGACION
7876	GIL	20	INVESTIGACION

EMP_NO	APELLIDO	DEP_NO	DEPARTAMENTO
7900	JIMENEZ	20	INVESTIGACION
6	CALVO	30	VENTAS
7499	ALONSO	30	VENTAS
7654	MARTIN	30	VENTAS
7698	GARRIDO	30	VENTAS
7844	CALVO	30	VENTAS
8998	CORTES	30	VENTAS

LEFT JOIN

```
SELECT EMP_NO, APELLIDO, D.DEP_NO, DNOMBRE
FROM VENTAS.EMPLEADOS E
    LEFT JOIN VENTAS.DEPARTAMENTOS D ON E.DEP_NO = D.DEP_NO;
```

EMP_NO	APELLIDO	DEP_NO	DEPARTAMENTO
1	REY	10	CONTABILIDAD
6	CALVO	30	VENTAS
38	GIL	20	INVESTIGACION
7499	ALONSO	30	VENTAS
7521	LOPEZ	10	CONTABILIDAD
7654	MARTIN	30	VENTAS
7698	GARRIDO	30	VENTAS
7782	MARTINEZ	10	CONTABILIDAD
7839	REY	10	CONTABILIDAD
7844	CALVO	30	VENTAS
7876	GIL	20	INVESTIGACION
7900	JIMENEZ	20	INVESTIGACION
7902	GARRIDO	NULL	NULL
8998	CORTES	30	VENTAS

RIGHT JOIN

```
SELECT EMP_NO, APELLIDO, D.DEP_NO, DNOMBRE
FROM VENTAS.EMPLEADOS E
    RIGHT JOIN VENTAS.DEPARTAMENTOS D ON E.DEP_NO = D.DEP_NO;
```

EMP_NO	APELLIDO	DEP_NO	DEPARTAMENTO
1	REY	10	CONTABILIDAD
7521	LOPEZ	10	CONTABILIDAD
7782	MARTINEZ	10	CONTABILIDAD
7839	REY	10	CONTABILIDAD
38	GIL	20	INVESTIGACION
7876	GIL	20	INVESTIGACION
7900	JIMENEZ	20	INVESTIGACION
6	CALVO	30	VENTAS
7499	ALONSO	30	VENTAS
7654	MARTIN	30	VENTAS
7698	GARRIDO	30	VENTAS
7844	CALVO	30	VENTAS
8998	CORTES	30	VENTAS
NULL	NULL	40	PRODUCCION