SQL
INTERVIEW QUESTIONS

Una colección de preguntas de SQL para preparar entrevistas técnicas

40
preguntas

ÍNDICE

Pregunta 1 ¿Qué es SQL y para qué se usa? 3
Pregunta 2 Explica las diferencias entre DDL, DML y DCL en SQL 4
Pregunta 3 ¿Qué son la clave primaria y la clave foránea? 5
Pregunta 4 Explica el concepto de normalización y sus ventajas 6
Pregunta 5 ¿Qué es JOIN y qué tipos de JOIN conoces? 7
Pregunta 6 ¿Qué es una subconsulta (subquery) y cuándo se usa? 8
Pregunta 7 ¿Cómo eliminar duplicados del resultado de una consulta SQL? 9
Pregunta 8 Explica la diferencia entre WHERE y HAVING 10
Pregunta 9 ¿Qué es un índice y cómo afecta al rendimiento? 11
Pregunta 10 ¿Cuáles son las principales diferencias entre DELETE y TRUNCATE? 12
Pregunta 11 ¿Qué es una transacción y qué propiedades tiene (ACID)? 13
Pregunta 12 ¿Qué son los triggers en SQL? 14
Pregunta 13 Explica qué es una VIEW y cuáles son sus ventajas 15
Pregunta 14 ¿Cómo se usa el operador LIKE y para qué sirve? 16
Pregunta 15 ¿Qué son las funciones de agregación? Pon ejemplos 17
Pregunta 16 Explica la diferencia entre UNION y UNION ALL 18
Pregunta 17 ¿Qué es un procedimiento almacenado y en qué se diferencia de una función? 19
Pregunta 18 ¿Cómo optimizar el rendimiento de las consultas SQL? 21
Pregunta 19 ¿Qué son las restricciones (constraints) y qué tipos existen? 22
Pregunta 20 ¿Qué es la inyección SQL y cómo protegerse de ella? 24
Pregunta 21 ¿Qué es una base de datos relacional? 26
Pregunta 22 Explica la diferencia entre INNER JOIN y OUTER JOIN 27
Pregunta 23 ¿Qué es NULL y cómo se trabaja con él en SQL? 29
Pregunta 24 ¿Cómo se usa la expresión CASE en SQL? 31
Pregunta 25 Explica las sentencias de transacción COMMIT y ROLLBACK 32
Pregunta 26 Explica las diferencias entre CHAR y VARCHAR 33
Pregunta 27 ¿Qué es una tabla temporal en SQL? 34
Pregunta 28 ¿Qué son las funciones de ventana en SQL? 35
Pregunta 29 Explica el concepto de CTE (Common Table Expression) 36
Pregunta 30 ¿Cómo se elimina una tabla junto con sus datos? 37
Pregunta 31 ¿Qué es FOREIGN KEY y cómo garantiza la integridad de los datos? 38
Pregunta 32 ¿Cómo se añade una columna nueva a una tabla existente? 39
Pregunta 33 ¿Qué es una subconsulta correlacionada? 40
Pregunta 34 Explica la diferencia entre DELETE, TRUNCATE y DROP 41
Pregunta 35 ¿Qué es un Self-Join y cuándo se usa? 43
Pregunta 36 ¿Cómo se hace una copia de seguridad de una base de datos y cómo se restaura? 44
Pregunta 37 ¿Cómo se implementa una relación de muchos a muchos en SQL? 46
Pregunta 38 ¿Cómo funcionan los comandos REVOKE y GRANT? 48
Pregunta 39 Explica el uso de las funciones hash en SQL 49
Pregunta 40 ¿Cómo funciona un trigger de SQL? 51

Pregunta 1

¿Qué es SQL y para qué se usa?

> *SQL (Structured Query Language) es el lenguaje estándar para interactuar con bases de datos relacionales. Se usa para definir, gestionar y extraer datos de las bases de datos. Con SQL puedes realizar las siguientes operaciones:*

- Crear nuevas bases de datos y tablas
- Insertar nuevos datos en las tablas
- Leer datos mediante consultas
- Actualizar datos existentes
- Eliminar datos
- Gestionar el acceso y los permisos

Pregunta 2

Explica las diferencias entre DDL, DML y DCL en SQL

Los comandos SQL se dividen en tres categorías principales:

DDL (Data Definition Language), lenguaje de definición de datos:

- Se usa para definir la estructura de la base de datos (el esquema).
- Comandos principales: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`, `RENAME`.

DML (Data Manipulation Language), lenguaje de manipulación de datos:

- Se usa para trabajar con los datos dentro de las tablas.
- Comandos principales: `SELECT`, `INSERT`, `UPDATE`, `DELETE`.

DCL (Data Control Language), lenguaje de control de datos:

- Se usa para gestionar los permisos de acceso a la base de datos.
- Comandos principales: `GRANT`, `REVOKE`.

Pregunta 3

¿Qué son la clave primaria y la clave foránea?

Clave primaria (PRIMARY KEY):

- Identificador único de un registro en la tabla.
- No admite duplicados ni valores `NULL`.
- Puede estar formada por una o varias columnas (clave compuesta).

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT
);
```

Clave foránea (FOREIGN KEY):

- Una columna o un conjunto de columnas que hacen referencia a la clave primaria de otra tabla.
- Garantiza la integridad referencial entre tablas.
- Permite relacionar registros de tablas distintas.

```
CREATE TABLE enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Pregunta 4

Explica el concepto de normalización y sus ventajas

> *La normalización es el proceso de organizar la estructura de una base de datos para reducir la redundancia de datos y garantizar su integridad.*

Objetivos principales de la normalización:

- Eliminar la redundancia: evita que los datos se dupliquen.
- Mejorar la integridad de los datos: minimiza la probabilidad de que los datos queden inconsistentes.
- Facilitar el mantenimiento y la actualización: hace que la estructura de la base de datos sea más flexible y comprensible.

Formas normales principales

- Primera forma normal (1FN): todas las columnas contienen valores atómicos (indivisibles).
- Segunda forma normal (2FN): cumple la 1FN y todas las columnas que no son clave dependen de la clave primaria completa.
- Tercera forma normal (3FN): cumple la 2FN y no hay dependencias transitivas entre columnas que no son clave.

Ejemplo

Antes de la normalización:

StudentID	StudentName	CourseID	CourseName
1	Juan	101	Matemáticas
1	Juan	102	Música

Después de la normalización:

Tabla Students

StudentID	StudentName
1	Juan

Tabla Courses

CourseID	CourseName
101	Matemáticas
102	Música

Tabla Enrollments

StudentID	CourseID
1	101
1	102

Tienes un análisis detallado del tema con ejemplos en el artículo «Normalización de bases de datos: las formas normales explicadas de forma sencilla».

Pregunta 5

¿Qué es JOIN y qué tipos de JOIN conoces?

> *JOIN es una operación de SQL que te permite combinar filas de dos o más tablas a partir de las columnas que las relacionan.*

Tipos de JOIN:

- `INNER JOIN`:

Devuelve los registros que tienen valores coincidentes en ambas tablas.

- `LEFT JOIN (o LEFT OUTER JOIN)`:

Devuelve todos los registros de la tabla izquierda y los registros coincidentes de la tabla derecha. Si no hay coincidencia, devuelve `NULL` para la tabla derecha.

- `RIGHT JOIN (o RIGHT OUTER JOIN)`:

Devuelve todos los registros de la tabla derecha y los registros coincidentes de la tabla izquierda. Si no hay coincidencia, devuelve `NULL` para la tabla izquierda.

- `FULL OUTER JOIN`:

Devuelve todos los registros cuando hay coincidencia en cualquiera de las tablas.

- `CROSS JOIN`:

Realiza el producto cartesiano de dos tablas, combinando cada fila de la primera tabla con cada fila de la segunda.

Pregunta 6

¿Qué es una subconsulta (subquery) y cuándo se usa?

> *Una subconsulta es una consulta SQL anidada dentro de otra consulta. Se usa para realizar operaciones cuyo resultado necesita la consulta principal.*

Casos de uso de las subconsultas:

- **Filtrado de datos**: usar los resultados de una subconsulta en las condiciones `WHERE` o `HAVING`.
- **Selección de datos**: usar una subconsulta en la lista de columnas seleccionadas.
- **Creación de tablas virtuales**: usar una subconsulta en la cláusula `FROM`.

Ejemplos:

1. Subconsulta en `WHERE`:

```
SELECT name
FROM employees
WHERE department_id = (SELECT id FROM departments WHERE name = 'IT');
```

2. Subconsulta en `FROM`:

```
SELECT sub.department, COUNT(*)
FROM (
    SELECT department_id AS department
    FROM employees
) sub
GROUP BY sub.department;
```

Pregunta 7

¿Cómo eliminar duplicados del resultado de una consulta SQL?

Usa la palabra clave `DISTINCT` en la sentencia `SELECT` para devolver solo registros únicos.

Ejemplo:

```
SELECT DISTINCT position
FROM employees;
```

Pregunta 8

Explica la diferencia entre WHERE y HAVING

WHERE

- Filtra filas **antes** de agrupar los datos.
- No puede usar funciones de agregación (`SUM()` , `COUNT()` , `AVG()` , etc.).
- Se aplica a registros individuales de la tabla.

Ejemplo:

```
SELECT department_id, COUNT()
FROM employees
WHERE salary > 50000
GROUP BY department_id;
```

HAVING

- *Filtra grupos de filas después de agrupar los datos.*
- *Puede usar funciones de agregación.*
- *Se aplica a los resultados de* `GROUP BY` .

Ejemplo:

```
SELECT department_id, COUNT() AS num_employees
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

Pregunta 9

¿Qué es un índice y cómo afecta al rendimiento?

> *Un **índice** es una estructura de datos especial que acelera las operaciones de lectura de datos de una tabla creando punteros a esos datos.*

Los índices aceleran las operaciones de lectura, pero pueden ralentizar las de escritura (inserción, actualización, eliminación), porque hay que actualizar los índices cada vez que cambian los datos.

Ventajas de los índices:

- Ejecución más rápida de las operaciones `SELECT`.
- Mejor rendimiento al ordenar y buscar datos.

Desventajas de los índices:

- Espacio adicional en disco.
- Operaciones `INSERT`, `UPDATE` y `DELETE` más lentas.

Ejemplo de creación de un índice:

```
-- Crear un índice sobre la columna name de la tabla employees
CREATE INDEX idx_employees_name ON employees(name);
```

Pregunta 10

¿Cuáles son las principales diferencias entre DELETE y TRUNCATE?

DELETE:

- Elimina los registros seleccionados de una tabla.
- Puedes usar la condición `WHERE` para eliminar registros concretos.
- La operación se registra fila a fila en el log de transacciones.
- Se disparan los triggers `ON DELETE`.
- Es más lenta que `TRUNCATE`.

Ejemplo:

```
DELETE FROM employees WHERE salary < 30000;
```

TRUNCATE:

- Elimina todos los registros de la tabla sin posibilidad de recuperarlos con `ROLLBACK` (en la mayoría de los SGBD).
- No se puede usar `WHERE`.
- La operación es más rápida porque no se registra fila a fila.
- Reinicia los identificadores (si se usa autoincremento).
- Los triggers no se disparan.

Ejemplo:

```
TRUNCATE TABLE employees;
```

Tienes un análisis detallado del tema con ejemplos en el artículo «TRUNCATE, DELETE y DROP: en qué se diferencian y cuál elegir».

Pregunta 11

¿Qué es una transacción y qué propiedades tiene (ACID)?

> *Una **transacción** es una secuencia de operaciones que se ejecutan como una única acción lógica: o se ejecuta por completo, o se deshace por completo.*

Propiedades de las transacciones (ACID):

Atomicidad (Atomicity):

- La transacción se ejecuta por completo o no se ejecuta en absoluto.
- Si se produce un fallo, se deshacen todos los cambios.

Consistencia (Consistency):

- La transacción lleva la base de datos de un estado consistente a otro.
- Se respetan todas las reglas y restricciones de la base de datos.

Aislamiento (Isolation):

- Los resultados de la transacción no son visibles para otras transacciones hasta que termina.
- Evita que las transacciones paralelas se influyan entre sí.

Durabilidad (Durability):

- Una vez que la transacción termina con éxito, sus resultados se conservan incluso si el sistema falla.
- Los cambios se escriben en un almacenamiento persistente.

Tienes un análisis detallado del tema con ejemplos en el artículo «ACID sin complicaciones: las 4 propiedades de las transacciones».

Pregunta 12

¿Qué son los triggers en SQL?

> *Un **trigger** es un procedimiento almacenado que se ejecuta automáticamente cuando se produce un determinado evento en la base de datos, como `INSERT`, `UPDATE` o `DELETE` sobre una tabla concreta.*

Tipos de triggers:

- Triggers DML: reaccionan a las operaciones `INSERT`, `UPDATE` y `DELETE`.
- Triggers DDL: reaccionan a las operaciones `CREATE`, `ALTER` y `DROP`.
- Triggers a nivel de fila o de sentencia.

Ventajas de los triggers:

- Automatización de comprobaciones y restricciones.
- Registro de los cambios.
- Mantenimiento de la integridad de los datos.

Ejemplo de creación de un trigger:

```
CREATE TRIGGER trg_after_insert_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_log (employee_id, action, action_time)
    VALUES (NEW.id, 'INSERT', NOW());
END;
```

Pregunta 13

Explica qué es una VIEW y cuáles son sus ventajas

> *Una **VIEW (vista)** es una tabla virtual basada en el resultado de una consulta SQL. La vista no almacena datos por sí misma, sino que ofrece una forma concreta de ver los datos de una o varias tablas.*

Ventajas de una VIEW:

- **Simplificar consultas complejas**: te permite guardar una consulta complicada y usarla como si fuera una tabla sencilla.
- **Seguridad**: da a los usuarios acceso solo a ciertos datos y oculta el resto.
- **Vistas actualizables**: en algunos casos se pueden actualizar los datos a través de la vista.
- **Mantenimiento de la integridad de los datos**: puede contener datos de varias tablas combinados de una forma determinada.

Ejemplo de creación de una VIEW:

```
CREATE VIEW employee_details AS
SELECT e.id, e.name, d.name AS department, e.salary
FROM employees e
JOIN departments d ON e.department_id = d.id;
```

Uso de una VIEW:

```
SELECT * FROM employee_details WHERE salary > 50000;
```

Pregunta 14

¿Cómo se usa el operador LIKE y para qué sirve?

El operador `LIKE` se usa en la cláusula `WHERE` para buscar cadenas que coincidan con un patrón determinado. En los patrones se usan caracteres comodín:

- `%`: coincide con cualquier secuencia de caracteres (incluida la de longitud cero).
- `_`: coincide con un único carácter cualquiera.

Ejemplos de uso:

1. Buscar cadenas que empiezan por «A»:

```
SELECT  FROM employees WHERE name LIKE 'A%';
```

2. Buscar cadenas que terminan en «e»:

```
SELECT  FROM employees WHERE name LIKE '%e';
```

3. Buscar cadenas cuyo segundo carácter es «a»:

```
SELECT * FROM employees WHERE name LIKE '_a%';
```

Pregunta 15

¿Qué son las funciones de agregación? Pon ejemplos

Las funciones de agregación realizan cálculos sobre un conjunto de valores y devuelven un único valor. A menudo se usan junto con `GROUP BY`.

Funciones de agregación principales

- `COUNT()`: cuenta el número de filas.
- `SUM()`: calcula la suma de los valores.
- `AVG()`: calcula el valor medio.
- `MAX()`: encuentra el valor máximo.
- `MIN()`: encuentra el valor mínimo.

Ejemplos de uso

1. Contar el número de empleados:

```
SELECT COUNT(*) FROM employees;
```

2. Salario medio por departamento:

```
SELECT department_id, AVG(salary) AS average_salary
FROM employees
GROUP BY department_id;
```

3. Salario máximo de la empresa:

```
SELECT MAX(salary) FROM employees;
```

4. Ventas totales del mes:

```
SELECT SUM(amount) FROM sales WHERE sale_date BETWEEN '2023-01-01' AND '2023-01-31';
```

Pregunta 16

Explica la diferencia entre UNION y UNION ALL

UNION:

- Combina los resultados de dos o más consultas `SELECT`.
- Elimina los duplicados del resultado combinado.

Sintaxis

```
SELECT column_list FROM table1
UNION
SELECT column_list FROM table2;
```

UNION ALL:

- Combina los resultados de dos o más consultas `SELECT`.
- Conserva los duplicados en el resultado combinado.
- Es más rápido, porque no hace la operación adicional de eliminar duplicados.

Sintaxis

```
SELECT column_list FROM table1
UNION ALL
SELECT column_list FROM table2;
```

Pregunta 17

¿Qué es un procedimiento almacenado y en qué se diferencia de una función?

Procedimiento almacenado (Stored Procedure):

- Conjunto de comandos SQL guardados en el servidor para reutilizarlos.
- Puede ejecutar operaciones `SELECT`, `INSERT`, `UPDATE` y `DELETE`.
- Puede devolver varios conjuntos de resultados o no devolver nada.
- Puede tener parámetros de entrada y de salida.
- No se puede llamar dentro de una consulta SQL.

Ejemplo de procedimiento almacenado:

```
CREATE PROCEDURE GetEmployeeByID(IN emp_id INT)
BEGIN
    SELECT * FROM employees WHERE id = emp_id;
END;
```

Llamada al procedimiento:

```
CALL GetEmployeeByID(1);
```

Función (Function):

- Devuelve un único valor (función escalar) o una tabla (función de tabla).
- Se puede usar en expresiones SQL (por ejemplo, en SELECT o WHERE).
- Tiene que devolver un valor.
- Normalmente se usa para cálculos y devuelve un resultado determinista.

Ejemplo de función:

```
CREATE FUNCTION GetEmployeeSalary(emp_id INT) RETURNS DECIMAL(10,2)
BEGIN
    DECLARE salary DECIMAL(10,2);
    SELECT e.salary INTO salary FROM employees e WHERE e.id = emp_id;
    RETURN salary;
END;
```

Uso de la función:

```
SELECT name, GetEmployeeSalary(id) FROM employees;
```

Pregunta 18

¿Cómo optimizar el rendimiento de las consultas SQL?

Usa índices:

- Crea índices sobre las columnas que aparecen a menudo en `WHERE`, `JOIN` y `ORDER BY`.
- Evita los índices redundantes.

**Evita `SELECT`*:

- **Selecciona solo las columnas que necesitas.**
- **Así se reduce el volumen de datos transferidos.**

Optimiza las condiciones de `JOIN` y `WHERE`:

- **Usa la igualdad (=) en lugar de la desigualdad siempre que puedas.**
- **Evita funciones y cálculos sobre columnas indexadas dentro de las condiciones.**

Limita los resultados (`LIMIT`):

- **Limita el número de filas devueltas si no necesitas todos los datos.**

Evita subconsultas cuando puedas usar un JOIN:

- **Sustituye las subconsultas correlacionadas por `JOIN` o `EXISTS`.**

Cachea los datos que se usan a menudo:

- **Usa vistas materializadas o caché a nivel de aplicación.**

Perfila y analiza las consultas:**

- Usa herramientas (`EXPLAIN`, `EXPLAIN PLAN`) para analizar el plan de ejecución de las consultas.

Tienes un análisis detallado del tema con ejemplos en el artículo «Optimización de consultas SQL: 10 formas de acelerar una consulta lenta».

Pregunta 19

¿Qué son las restricciones (constraints) y qué tipos existen?

Las **restricciones** garantizan la integridad y la fiabilidad de los datos de una tabla definiendo reglas para los valores de las columnas.

Tipos de restricciones:

NOT NULL:

- Prohíbe guardar valores `NULL` en la columna.

Ejemplo:

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```

UNIQUE:

- Garantiza que los valores de una columna o de un grupo de columnas sean únicos.

Ejemplo:

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

PRIMARY KEY:

- Combinación de `NOT NULL` y `UNIQUE`.
- Identifica cada registro de la tabla.

FOREIGN KEY:

- Garantiza la integridad referencial entre tablas.
- El valor tiene que coincidir con un valor existente de la clave primaria de la tabla relacionada.

Ejemplo:

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    user_id INT,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

CHECK:

- Define una condición que deben cumplir los valores de la columna.

Ejemplo:

```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    age INT CHECK (age >= 18)
);
```

DEFAULT:

- Establece un valor por defecto para la columna si no se indica ninguno al insertar.

```
CREATE TABLE tasks (
    task_id INT PRIMARY KEY,
    status VARCHAR(20) DEFAULT 'Pending'
);
```

Pregunta 20

¿Qué es la inyección SQL y cómo protegerse de ella?

> *La **inyección SQL** es un método de ataque a una base de datos en el que el atacante introduce código SQL malicioso a través de los datos de entrada y consigue ejecutar consultas SQL no autorizadas.*

Consecuencias de una inyección SQL:

- Robo de datos.
- Eliminación o modificación de datos.
- Obtención de acceso de administrador.

Formas de protegerse de las inyecciones SQL:

1. Consultas parametrizadas (Prepared Statements):

- Usar parámetros en lugar de concatenar cadenas.
- El SGBD escapa automáticamente los caracteres especiales.

Ejemplo en Java con JDBC:

```
String sql = "SELECT * FROM users WHERE username = ? AND password = ?";
PreparedStatement stmt = connection.prepareStatement(sql);
stmt.setString(1, username);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

2. Usar un ORM (Object-Relational Mapping):

- Las bibliotecas ORM suelen incluir mecanismos de protección frente a inyecciones SQL.

3. Validar y filtrar los datos de entrada:

- Comprobar que los datos tienen el formato esperado.
- Validar tanto en el servidor como en el cliente.

4. Restringir los permisos de acceso:

- Dar a los usuarios de la base de datos solo los permisos mínimos necesarios.
- Restringir el acceso a las tablas y operaciones del sistema.

5. Usar procedimientos almacenados:

- La lógica de trabajo con los datos queda encapsulada en el procedimiento.

- Los usuarios solo tienen acceso a los procedimientos, no a las tablas directamente.

Pregunta 21

¿Qué es una base de datos relacional?

> *Una **base de datos relacional** es una base de datos basada en el modelo relacional de datos. En ella, los datos se guardan en tablas y las relaciones entre ellos se definen mediante claves.*

Características principales:

- Tablas (relaciones): los datos se organizan en tablas formadas por filas y columnas.
- Filas (registros): cada fila representa un registro independiente.
- Columnas (atributos): cada columna contiene datos de un tipo determinado.
- Claves: sirven para identificar los registros y establecer relaciones entre tablas.

Ventajas de las bases de datos relacionales:

- Flexibilidad: es fácil añadir nuevas tablas y columnas.
- Integridad de los datos: se usan restricciones para mantener la integridad.
- SQL: un lenguaje estándar para gestionar los datos.

Pregunta 22

Explica la diferencia entre INNER JOIN y OUTER JOIN

INNER JOIN:

- Devuelve solo los registros que tienen valores coincidentes en las dos tablas combinadas.
- Si no hay coincidencia, el registro no entra en el resultado.

Ejemplo:

```
SELECT
FROM employees e
INNER JOIN departments d ON e.department_id = d.id;
```

OUTER JOIN:

- *Devuelve los registros que tienen coincidencias y también los registros de una de las tablas que no las tienen.*

Tipos de OUTER JOIN:

LEFT OUTER JOIN (LEFT JOIN):

- *Devuelve todos los registros de la tabla izquierda y los registros coincidentes de la tabla derecha.*
- *Si no hay coincidencia, las columnas de la tabla derecha serán* `NULL`.

Ejemplo:

```
SELECT
FROM employees e
LEFT JOIN departments d ON e.department_id = d.id;
```

RIGHT OUTER JOIN (RIGHT JOIN):

- Devuelve todos los registros de la tabla derecha y los registros coincidentes de la tabla izquierda.
- Si no hay coincidencia, las columnas de la tabla izquierda serán `NULL`.

Ejemplo:

```
SELECT
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.id;
```

FULL OUTER JOIN (FULL JOIN):

- *Devuelve todos los registros cuando hay coincidencia en cualquiera de las tablas.*
- *Si no hay coincidencia, las columnas correspondientes serán* `NULL`.

```
SELECT
FROM employees e
FULL OUTER JOIN departments d ON e.department_id = d.id;
```

Pregunta 23

¿Qué es NULL y cómo se trabaja con él en SQL?

> ***NULL** es un valor especial de SQL que indica la ausencia de datos o un valor desconocido.*

Particularidades de NULL:

- `NULL` no equivale a una cadena vacía ni a cero.
- La aritmética, las comparaciones y la mayoría de las funciones con `NULL` devuelven `NULL`: `5 + NULL` es `NULL`.
- La comparación `NULL = NULL` también devuelve `NULL`, no `true`: no se sabe si dos valores desconocidos son iguales. Por eso `WHERE campo = NULL` no encuentra ninguna fila; para comprobarlo necesitas `IS NULL`.
- Los operadores lógicos son la excepción: `NULL OR true` devuelve `true` y `NULL AND false` devuelve `false`.

Trabajar con NULL

Para comprobar si un valor es `NULL` se usa el operador `IS NULL` o `IS NOT NULL`.

```
-- Buscar registros con fecha de nacimiento desconocida
SELECT  FROM employees WHERE birth_date IS NULL;

-- Buscar registros con fecha de nacimiento conocida
SELECT  FROM employees WHERE birth_date IS NOT NULL;
```

Funciones para trabajar con NULL

COALESCE Devuelve el primer elemento de la lista que no sea `NULL`.

```
COALESCE(val1[, val2, ...., val_n])
```

ISNULL Devuelve 1 o 0 según la expresión sea o no `NULL`.

```
ISNULL(value)
```

IFNULL Devuelve el valor pasado como primer argumento si no es `NULL`. En caso contrario, devuelve el valor pasado como segundo argumento.

```
IFNULL(value, alternative_value)
```

Tienes un análisis detallado del tema con ejemplos en el artículo «COALESCE en SQL: sustituir NULL y combinarlo con NULLIF».

Pregunta 24

¿Cómo se usa la expresión CASE en SQL?

La expresión `CASE` se usa para implementar lógica condicional en las consultas SQL. Te permite devolver valores en función de condiciones, de forma parecida a `IF-ELSE`.

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE default_result
END
```

Ejemplos

- Asignar categorías según el salario:

```
SELECT name,
    salary,
    CASE
        WHEN salary >= 80000 THEN 'Alto'
        WHEN salary >= 50000 THEN 'Medio'
        ELSE 'Bajo'
    END AS salary_category
FROM employees;
```

Pregunta 25

Explica las sentencias de transacción COMMIT y ROLLBACK

COMMIT

- Confirma la transacción actual.
- Todos los cambios hechos en la transacción pasan a ser permanentes y visibles para otros usuarios.
- Después de COMMIT ya no se pueden deshacer los cambios.

```
BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

COMMIT;
```

ROLLBACK

- Cancela la transacción actual.
- Se deshacen todos los cambios hechos en la transacción.
- La base de datos vuelve al estado que tenía antes de empezar la transacción.

```
BEGIN TRANSACTION;

DELETE FROM orders WHERE order_date < '2022-01-01';

-- Si cambias de opinión
ROLLBACK;
```

Uso en la gestión de transacciones:

- `BEGIN TRANSACTION` o `START TRANSACTION` : inicia la transacción.
- `COMMIT` : confirma la transacción.
- `ROLLBACK` : deshace la transacción.

Pregunta 26

Explica las diferencias entre CHAR y VARCHAR

CHAR(n):

- Guarda cadenas de longitud fija `n`.
- Si la cadena introducida es más corta que `n`, se rellena con espacios hasta alcanzar la longitud `n`.
- Se usa para datos de la misma longitud (por ejemplo, códigos de país o códigos postales).

VARCHAR(n):

- Guarda cadenas de longitud variable de hasta `n` caracteres.
- En la práctica ocupa tanto espacio como caracteres tiene la cadena, más un pequeño espacio adicional para guardar la longitud.
- Se usa para datos de texto de longitud variable.

Diferencias principales:

Memoria y rendimiento:

- `CHAR` siempre ocupa una cantidad fija de memoria.
- `VARCHAR` es más eficiente en el uso de memoria, pero puede ser un poco más lento en el acceso.

Uso:

- `CHAR` es adecuado para datos con una longitud predecible.
- `VARCHAR` es adecuado para datos de longitud variable.

Ejemplo:

```
CREATE TABLE products (
    code CHAR(10),     -- Código de producto de longitud fija
    name VARCHAR(100)  -- Nombre de producto de longitud variable
);
```

Inserción de datos:

```
INSERT INTO products (code, name)
VALUES ('A123', 'Portátil Lenovo');
```

En la columna `code`, el valor se rellenará con espacios hasta los 10 caracteres.

Pregunta 27

¿Qué es una tabla temporal en SQL?

> *Una **tabla temporal** es una tabla que solo existe dentro de la sesión o conexión actual y se elimina automáticamente cuando termina la sesión o se cierra la conexión.*

Crear una tabla temporal

```
CREATE TEMPORARY TABLE TempTable (
    id INT,
    name VARCHAR(100)
);
```

Usar una tabla temporal

```
-- Insertar datos en la tabla temporal
INSERT INTO #TempTable (id, name)
VALUES (1, 'Iván'), (2, 'Pedro');

-- Leer datos de la tabla temporal
SELECT * FROM #TempTable;

-- La tabla temporal se eliminará automáticamente al terminar la sesión
```

Para qué se usa una tabla temporal

- Guardar resultados intermedios en consultas complejas.
- Procesar grandes volúmenes de datos en operaciones por lotes.
- Evitar conflictos cuando varios usuarios trabajan a la vez.

Pregunta 28

¿Qué son las funciones de ventana en SQL?

> *Las **funciones de ventana** son funciones que realizan cálculos sobre un conjunto de filas (la ventana) relacionadas con la fila actual y devuelven un resultado para cada fila sin agrupar los datos.*

Funciones de ventana principales:

Funciones de agregación:

- `SUM` : calcula la suma total de los valores
- `COUNT` : cuenta el número total de registros de la columna
- `AVG` : calcula la media aritmética
- `MAX` : encuentra el valor máximo
- `MIN` : encuentra el valor mínimo

Funciones de ranking:

- `ROW_NUMBER` : asigna a la fila un número consecutivo dentro de la ventana
- `RANK` : asigna a la fila un puesto dentro de la ventana, con saltos cuando hay valores iguales
- `DENSE_RANK` : asigna a la fila un puesto sin saltos

Funciones de desplazamiento:

- `LAG` : devuelve el valor de la fila anterior
- `LEAD` : devuelve el valor de la fila siguiente
- `FIRST_VALUE` : devuelve el primer valor de la ventana
- `LAST_VALUE` : devuelve el último valor de la ventana

Tienes una explicación detallada de cómo funcionan las funciones de ventana en nuestro curso.

Tienes un análisis detallado del tema con ejemplos en el artículo «ROW_NUMBER, RANK y DENSE_RANK: las diferencias con un solo ejemplo».

Pregunta 29

Explica el concepto de CTE (Common Table Expression)

> *Una **CTE (Common Table Expression)** es un conjunto de resultados temporal con nombre, definido dentro de una consulta SQL con la palabra clave `WITH`.*

Mejora la legibilidad y la estructura de las consultas complejas.

Sintaxis

```
WITH CTEName (column1, column2, ...)
AS (
    -- Tu consulta
    SELECT ...
)
SELECT * FROM CTEName;
```

Ventajas de las CTE

- Mejoran la legibilidad del código.
- Permiten dividir las consultas complejas en partes lógicas.
- Admiten consultas recursivas (CTE recursivas).

Tienes una explicación detallada de cómo funcionan las CTE en nuestro curso.

Tienes un análisis detallado del tema con ejemplos en el artículo «CTE en SQL: qué es una Common Table Expression, con recursión».

Pregunta 30

¿Cómo se elimina una tabla junto con sus datos?

Para eso se usa el comando `DROP TABLE`, que elimina la tabla y todos sus datos de la base de datos.

Sintaxis

```
DROP TABLE table_name;
```

Particularidades:

- Se eliminan todos los datos, índices, triggers y permisos asociados a la tabla.
- La acción es irreversible si no hay copias de seguridad o mecanismos de recuperación configurados.

Pregunta 31

¿Qué es FOREIGN KEY y cómo garantiza la integridad de los datos?

> *Una **FOREIGN KEY** (clave foránea) es una restricción que establece una relación entre una columna o un conjunto de columnas de una tabla y una o varias columnas de otra tabla (normalmente, su clave primaria).*

Garantiza la integridad referencial, asegurando que los valores de la columna de clave foránea coincidan con valores existentes en la tabla relacionada.

Cómo se garantiza la integridad de los datos:

- **Se impide insertar datos incorrectos.**

No puedes insertar un valor en la columna de clave foránea si ese valor no existe en la tabla relacionada.

- **Se restringe la eliminación de registros relacionados.**

No puedes eliminar un registro de la tabla padre si hay registros de la tabla hija que hacen referencia a él, salvo que tomes medidas adicionales.

Ejemplo de clave foránea:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

Pregunta 32

¿Cómo se añade una columna nueva a una tabla existente?

Para eso se usa la sentencia `ALTER TABLE` con la cláusula `ADD`, que añade una columna nueva a la tabla.

Sintaxis

```
ALTER TABLE table_name
ADD column_name data_type [constraints];
```

Ejemplo

Supongamos que tenemos la tabla `employees` y queremos añadir la columna `email` de tipo `VARCHAR(255)`.

```
ALTER TABLE employees
ADD email VARCHAR(255);
```

Añadir una columna con la restricción `NOT NULL` y un valor por defecto:

```
ALTER TABLE employees
ADD date_of_birth DATE NOT NULL DEFAULT '1900-01-01';
```

Particularidad Si añades una columna con la restricción `NOT NULL` y la tabla ya contiene datos, tienes que indicar un valor por defecto; si no, obtendrás un error.

Pregunta 33

¿Qué es una subconsulta correlacionada?

> *Una **subconsulta correlacionada** es una subconsulta que depende de la consulta externa. Se ejecuta para cada fila de la consulta externa usando los valores de esa fila.*

Particularidades

- La subconsulta hace referencia a columnas de la consulta externa.
- Puede ser menos eficiente porque se ejecuta muchas veces.

Ejemplo

Tenemos las tablas `employees` y `departments`.

```
SELECT e.name, e.salary
FROM employees e
WHERE e.salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department_id = e.department_id
);
```

- Aquí la subconsulta calcula el salario medio del departamento de cada empleado.
- La consulta principal selecciona a los empleados cuyo salario supera la media de su departamento.

Tienes más información sobre las subconsultas correlacionadas en nuestro curso.

Pregunta 34

Explica la diferencia entre DELETE, TRUNCATE y DROP

DELETE

- Elimina los registros seleccionados de una tabla.
- Puedes usar la condición `WHERE` para eliminar registros concretos.
- La operación se registra fila a fila en el log de transacciones.
- Se disparan los triggers `ON DELETE`.

Sintaxis

```
DELETE FROM table_name [WHERE condition];
```

TRUNCATE

- Elimina todos los datos de la tabla sin posibilidad de recuperarlos.
- No se puede usar `WHERE`.
- Es más rápido porque no registra la eliminación de cada fila.
- Reinicia los identificadores de autoincremento.
- Los triggers no se disparan.

Sintaxis

```
TRUNCATE TABLE table_name;
```

DROP

- Elimina la tabla entera junto con sus datos, estructura, índices y restricciones.
- La acción es irreversible.

Sintaxis

```
DROP TABLE table_name;
```

Cómo elegir entre los comandos

- `DELETE` se usa cuando necesitas eliminar registros concretos.

- `TRUNCATE` se usa para eliminar rápidamente todos los datos de una tabla conservando su estructura.
- `DROP` se usa para eliminar por completo una tabla de la base de datos.

Tienes un análisis detallado del tema con ejemplos en el artículo «TRUNCATE, DELETE y DROP: en qué se diferencian y cuál elegir».

Pregunta 35

¿Qué es un Self-Join y cuándo se usa?

> *Un **Self-Join** es un tipo de combinación en SQL en el que una tabla se combina consigo misma. Es útil cuando necesitas comparar filas de una misma tabla entre sí o procesar datos jerárquicos.*

Cuándo se usa un Self-Join:

- **Estructuras jerárquicas.**

Por ejemplo, en una tabla de empleados donde cada empleado puede tener un responsable que también es empleado y está en la misma tabla.

- **Comparación de registros.**

Para encontrar duplicados o comparar valores entre distintas filas de una misma tabla.

Ejemplo

employeeId	name	managerId
1	John	
2	Michail	1
3	Alisa	1
4	Max	2

Para obtener la lista de empleados con sus responsables:

```
SELECT e.name AS Employee, m.name AS Manager
FROM Employee e
LEFT JOIN Employee m ON e.managerId = m.employeeId;
```

En esta consulta combinamos la tabla `Employee` consigo misma para emparejar a cada empleado con su responsable.

Pregunta 36

¿Cómo se hace una copia de seguridad de una base de datos y cómo se restaura?

Hacer copias de seguridad de una base de datos y restaurarlas son operaciones críticas para mantener los datos a salvo y poder recuperarlos en caso de fallo o pérdida.

Cómo hacer una copia de seguridad de una base de datos:

Los métodos de copia de seguridad dependen del sistema de gestión de bases de datos (SGBD) que uses.

A continuación tienes ejemplos para algunos SGBD populares.

MySQL

Copia de seguridad con la utilidad `mysqldump`:

```
mysqldump -u username -p mydatabase > backup.sql
```

- `username`: nombre de usuario de la base de datos.
- `mydatabase`: nombre de la base de datos que quieres copiar.
- `backup.sql`: archivo en el que se guardará la copia de seguridad.

PostgreSQL

Copia de seguridad con la utilidad `pg_dump`:

```
pg_dump -U username mydatabase > backup.sql
```

- `-U username`: nombre de usuario de la base de datos.
- `mydatabase`: nombre de la base de datos.
- `backup.sql`: archivo de salida de la copia de seguridad.

Cómo restaurar una base de datos a partir de una copia de seguridad:

MySQL

Restaurar la base de datos a partir del archivo `backup.sql`:

```
mysql -u username -p mydatabase < backup.sql
```

PostgreSQL

Restaurar la base de datos con la utilidad `psql`:

```
psql -U username mydatabase < backup.sql
```

Recomendaciones

- **Permisos de acceso**

Asegúrate de tener los permisos necesarios para hacer copias de seguridad y restauraciones.

- **Periodicidad**

Configura copias de seguridad automáticas y periódicas para minimizar el riesgo de perder datos.

- **Almacenamiento de las copias**

Guarda las copias de seguridad en un lugar seguro y fiable, preferiblemente fuera del servidor principal.

- **Pruebas**

Comprueba de vez en cuando las copias de seguridad restaurándolas en un servidor de pruebas para asegurarte de que están íntegras y funcionan.

Pregunta 37

¿Cómo se implementa una relación de muchos a muchos en SQL?

Las relaciones de **muchos a muchos** en las bases de datos relacionales aparecen cuando un registro de la primera tabla puede corresponder a varios registros de la segunda, y viceversa.

En SQL, estas relaciones se implementan con una tabla intermedia (también llamada tabla de relación o tabla de unión), que conecta las dos tablas principales mediante claves foráneas.

Cómo implementar una relación de muchos a muchos en SQL

- **Crea las dos tablas principales** que quieres relacionar.
- **Crea una tabla intermedia** con claves foráneas que apunten a las claves primarias de ambas tablas principales.
- **Define las claves foráneas y una clave primaria compuesta** en la tabla intermedia para garantizar la integridad referencial y que cada par de la relación sea único.

Ejemplo de implementación

Imagina un escenario con las tablas `Student` (estudiantes) y `Course` (cursos), en el que un estudiante puede matricularse en varios cursos y un curso puede tener varios estudiantes.

- Crear la tabla de estudiantes:

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100)
);
```

- Crear la tabla de cursos:

```
CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    Title VARCHAR(100)
);
```

- Crear la tabla intermedia para establecer la relación de muchos a muchos:

```
CREATE TABLE StudentCourse (
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

Resultado:

- La **tabla intermedia** `StudentCourse` contiene pares de `StudentID` y `CourseID` que representan las relaciones entre estudiantes y cursos.
- La **clave primaria compuesta** (`StudentID` , `CourseID`) garantiza que cada par sea único y evita relaciones duplicadas.
- Las **claves foráneas** garantizan la integridad de los datos al hacer referencia a los registros correspondientes de las tablas `Student` y `Course` .

Pregunta 38

¿Cómo funcionan los comandos REVOKE y GRANT?

Los comandos `GRANT` y `REVOKE` en SQL se usan para gestionar los permisos de acceso de los usuarios a los objetos de la base de datos. Te permiten conceder o revocar privilegios concretos a usuarios o roles, lo que garantiza la seguridad y el control sobre quién puede hacer qué en la base de datos.

GRANT

El comando `GRANT` concede a usuarios o roles determinados privilegios sobre los objetos de la base de datos.

Sintaxis:

```
GRANT privileges ON object TO user [WITH GRANT OPTION];
```

- `privileges` : las acciones permitidas (por ejemplo, SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES).
- `object` : base de datos, tabla, vista, procedimiento, etc.
- `user` : nombre del usuario o del rol al que se conceden los permisos.
- `WITH GRANT OPTION` (opcional): permite a quien recibe los privilegios transmitirlos a otros usuarios.

Ejemplo: Así puedes conceder al usuario `user1` permiso para seleccionar datos de la tabla `employees` :

```
GRANT SELECT ON employees TO user1;
```

REVOKE

El comando `REVOKE` retira a usuarios o roles los privilegios concedidos anteriormente.

Sintaxis:

```
REVOKE privileges ON object FROM user;
```

Ejemplo: Retirar al usuario `user1` el permiso para seleccionar datos de la tabla `employees` :

```
REVOKE SELECT ON employees FROM user1;
```

Pregunta 39

Explica el uso de las funciones hash en SQL

Las funciones hash en SQL se usan para convertir datos de entrada de longitud arbitraria en una cadena de longitud fija. Esta conversión se llama **hashing** y se usa mucho para garantizar la seguridad y la integridad de los datos y para optimizar las operaciones de comparación y búsqueda.

Principales usos de las funciones hash en SQL:

1. Almacenamiento de contraseñas

- En lugar de guardar las contraseñas en texto plano, se guardan como hashes. Esto aumenta la seguridad, porque incluso si la base de datos se ve comprometida, los atacantes no podrán recuperar las contraseñas originales.
- Se usan funciones hash criptográficas como `SHA-256`, `SHA-512` y otras.

2. Comprobación de la integridad de los datos

Las funciones hash permiten saber si los datos se han modificado. Si calculas el hash de los datos originales y lo comparas con el hash actual, puedes detectar los cambios.

3. Indexación y optimización de búsquedas

Los valores hash pueden usarse para comparar rápidamente grandes volúmenes de datos o para crear índices que aceleren las búsquedas.

4. Generación de identificadores únicos

El hashing ayuda a generar identificadores únicos para los registros a partir de su contenido.

Ejemplo de uso de funciones hash en SQL:

```
-- Creamos la tabla de usuarios
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    PasswordHash VARBINARY(512) NOT NULL
);

-- Al añadir un usuario nuevo, calculamos el hash de su contraseña
INSERT INTO Users (UserID, Username, PasswordHash)
VALUES (1, 'user1', HASHBYTES('SHA2_512', 'password123'));

-- Comprobamos si la contraseña introducida coincide con el hash de la base de datos
SELECT UserID FROM Users
WHERE Username = 'user1' AND PasswordHash = HASHBYTES('SHA2_512', 'password123');
```

Recomendaciones y particularidades

Elección de la función hash

- Da preferencia a algoritmos más modernos y robustos, como `SHA-256` o `SHA-512`.
- Evita funciones obsoletas como `MD5` o `SHA1`, porque tienen vulnerabilidades conocidas.

Almacenamiento seguro

Guarda los hashes en formato binario (`VARBINARY`) en lugar de como cadena (`VARCHAR`) para conservar la precisión y ahorrar espacio.

El hash es irreversible

Recuerda que el hashing es una función unidireccional: no es posible recuperar los datos originales a partir del hash.

Pregunta 40

¿Cómo funciona un trigger de SQL?

Qué es un trigger de SQL

> *Un **trigger de SQL** es un objeto de la base de datos, un tipo especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un determinado evento en la base de datos. Estos eventos pueden ser operaciones `INSERT`, `UPDATE` o `DELETE` sobre tablas o vistas.*

Cómo funciona un trigger de SQL

El trigger se activa automáticamente en respuesta a un evento definido (por ejemplo, cuando se añade un registro nuevo a una tabla).

Tipos de triggers

- `BEFORE`: se dispara antes de ejecutar la operación.

Se usa para comprobar o modificar los datos antes de guardarlos.

- `AFTER`: se dispara después de ejecutar la operación.

Suele usarse para registrar cambios o actualizar tablas relacionadas.

- `INSTEAD OF`: sustituye el comportamiento estándar de la operación.

Se usa cuando hay que redefinir la acción por defecto, por ejemplo, al trabajar con vistas.

Dentro del trigger puedes acceder a los valores antiguos y nuevos de los datos mediante las pseudotablas especiales `OLD` y `NEW`.

Ejemplo de uso de un trigger

Tenemos la tabla `Employees` y queremos guardar automáticamente el historial de cambios de salario de los empleados en la tabla `SalaryHistory` cada vez que se actualizan los datos.

Creación de las tablas:

```
-- Tabla de empleados
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Salary DECIMAL(10, 2)
);

-- Tabla del historial de salarios
CREATE TABLE SalaryHistory (
    EmployeeID INT,
    OldSalary DECIMAL(10, 2),
    NewSalary DECIMAL(10, 2),
    ChangeDate DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Creación del trigger:

```
DELIMITER $$

CREATE TRIGGER trg_AfterSalaryUpdate
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
    IF OLD.Salary <> NEW.Salary THEN
        INSERT INTO SalaryHistory (EmployeeID, OldSalary, NewSalary)
        VALUES (NEW.EmployeeID, OLD.Salary, NEW.Salary);
    END IF;
END$$

DELIMITER ;
```

Cuándo usar triggers

- **Registro y auditoría:**

Registrar automáticamente los cambios en los datos para hacer un seguimiento de las acciones de los usuarios.

- **Integridad de los datos:**

Aplicar reglas de negocio complejas que no se pueden implementar con restricciones.

- **Sincronización de datos:**

Actualizar o modificar automáticamente las tablas relacionadas cuando cambian los datos.

- **Cálculo de valores:**

Calcular y actualizar automáticamente datos agregados o derivados.