☑ 6. SQL — 50 preguntas técnicas

1. Diferencia entre INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL JOIN

- **INNER JOIN**: devuelve solo registros con coincidencia en ambas tablas.
- **LEFT JOIN**: devuelve todos los registros de la tabla izquierda + coincidencias de la derecha.
- **RIGHT JOIN**: devuelve todos los registros de la tabla derecha + coincidencias de la izquierda.
- **FULL JOIN**: devuelve todos los registros cuando hay coincidencias en cualquiera de las dos tablas.

---

2. Clave primaria y clave foránea

- **Primary Key**: identifica un registro de forma única; no permite NULL ni duplicados.
- **Foreign Key**: referencia la PK de otra tabla; mantiene integridad referencial.

---

3. ¿Qué es una subconsulta y cuándo usarla?

Una consulta dentro de otra.
Útil para: filtros, cálculos intermedios, validación de valores dependientes o comparaciones.

---

4. Subconsulta correlacionada vs no correlacionada

- **Correlacionada**: depende de la fila externa (se ejecuta por cada fila).
- **No correlacionada**: se ejecuta una sola vez y su resultado se usa en la consulta externa.

---

5. ¿Qué es un índice y cómo mejora el rendimiento?

Estructura (árbol B o bitmap) que acelera búsquedas, ordenamientos y filtros.
Reduce lecturas de disco.

---

6. Desventajas de un índice

- Ocupan espacio.
- Ralentizan INSERT, UPDATE y DELETE.
- Pueden desactualizarse si no se mantienen.

---

7. ¿Qué es un índice compuesto?

Índice sobre múltiples columnas. Útil cuando las consultas filtran por más de una columna.

---

8. ¿Qué es una vista y para qué sirve?

Consulta almacenada como objeto lógico.
Sirve para:

- Simplificar SQL complejo
- Seguridad
- Abstracción de datos

---

9. Vista normal vs materializada

- **Vista normal**: se calcula al consultar.
- **Vista materializada**: almacena datos físicos y se refresca periódicamente.

---

10. Normalización (1NF, 2NF, 3NF)

- **1NF**: sin valores repetidos ni multivaluados; cada columna con datos atómicos.
- **2NF**: cumple 1NF y no hay dependencias parciales en PK compuesta.
- **3NF**: cumple 2NF y no hay dependencias transitivas.

---

11. Desnormalización

Proceso contrario: duplicar datos para acelerar consultas.
Útil en analítica y sistemas de alta lectura.

---

12. DELETE vs TRUNCATE vs DROP

- **DELETE**: borra filas → registra en log, permite WHERE.
- **TRUNCATE**: borra *todas* las filas → rápido, no usa log completo.
- **DROP**: elimina la *tabla* completa.

---

13. MERGE

Permite **INSERT/UPDATE/DELETE** según coincidencias.

```
MERGE INTO clientes c
USING staging s ON c.id = s.id
WHEN MATCHED THEN UPDATE SET c.nombre = s.nombre
WHEN NOT MATCHED THEN INSERT (id, nombre) VALUES (s.id, s.nombre);
```

---

14. Plan de ejecución

Desglose del motor SQL sobre cómo ejecutará una consulta.

---

15. ¿Cómo interpretar EXPLAIN?

Analizar:

- Orden de operaciones
- Tipo de join
- Uso de índices
- Costos estimados
- Lecturas (scans)

---

16. Tabla particionada

Tabla dividida en partes (particiones) según rango, lista o hash.

---

17. Tipos de particiones

- **RANGE**
- **LIST**
- **HASH**
- **COMPOSITE** (mix)

---

18. Sharding

Particionar datos en múltiples servidores físicos.

---

19. Índice bitmap

Usado en columnas con baja cardinalidad; eficiente en consultas analíticas.

---

20. Índice hash

Ideal para igualdad (=). Común en motores como PostgreSQL o DynamoDB.

---

21. GROUP BY

Agrupa filas por una o varias columnas.
Reglas:

- Columnas no agregadas deben estar en el GROUP BY.

---

22. HAVING

Filtro posterior al GROUP BY; se usa con agregaciones.

---

23. COUNT(*), COUNT(1), COUNT(columna)

- **COUNT(*)** = cuenta todas las filas.
- **COUNT(1)** = igual que COUNT(*).
- **COUNT(col)** = solo valores NO nulos.

---

24. Funciones window

Permiten cálculos sobre un conjunto de filas sin agrupar.
Ej: `SUM() OVER()`, `AVG() OVER(PARTITION BY ...)`.

---

25. ROW_NUMBER vs RANK vs DENSE_RANK

- **ROW_NUMBER**: sin empates.
- **RANK**: empates → saltos en numeración.
- **DENSE_RANK**: empates → sin saltos.

---

26. LAG y LEAD

Acceden al valor previo (LAG) o siguiente (LEAD) dentro de una ventana.

---

27. Self join

Join de una tabla consigo misma.

---

28. CTE

Consulta temporal nombrada usando `WITH`.

---

29. WITH recursivo vs no recursivo

- **No recursivo**: una sola consulta temporal.
- **Recursivo**: se llama a sí mismo (árboles, jerarquías).

---

30. Transacción

Conjunto de operaciones que se ejecutan como unidad atómica.

---

31. Niveles de aislamiento

- **READ UNCOMMITTED**: sucio.
- **READ COMMITTED**: sin lecturas sucias.
- **REPEATABLE READ**: sin lecturas no repetibles.
- **SERIALIZABLE**: mayor aislamiento.

---

32. Lock

Bloqueo de filas o tablas para evitar inconsistencias.

---

33. Deadlock vs blocking

- **Blocking**: una transacción espera a otra.
- **Deadlock**: dos transacciones se bloquean mutuamente.

---

34. Triggers

Acciones automáticas ante eventos (INSERT/UPDATE/DELETE).

---

35. Problemas con triggers

- Difícil debugging
- Rendimiento
- Dependencias ocultas

---

36. ACID

- **Atomicity**
- **Consistency**
- **Isolation**
- **Durability**

Garantizan confiabilidad de transacciones.

---

37. Optimizar consulta lenta

- Agregar índices adecuados
- Evitar funciones en columnas indexadas
- Reescribir joins
- Analizar plan de ejecución

---

38. Cardinalidad

Cantidad de valores distintos de una columna.
Impacta en elección de índices.

---

39. Join algorithms

- **Nested loop**: bueno para pocos registros + índice.
- **Hash join**: bueno para igualdad en grandes volúmenes.
- **Merge join**: tablas ordenadas o con índices.

---

40. Table scan vs index scan

- **Table scan**: lee toda la tabla.

- **Index scan**: usa índice; más rápido.

---

41. Función agregada

SUM, AVG, MIN, MAX, COUNT.

---

42. Función escalar

Opera sobre una sola fila: UPPER(), ROUND(), SUBSTR().

---

43. Problemas con funciones en columnas indexadas

Rompen el uso del índice → provocan table scans.

---

44. Validar duplicados

```
SELECT col, COUNT(*)
FROM tabla
GROUP BY col
HAVING COUNT(*) > 1;
```

---

45. Claves naturales vs sustitutas

- **Natural**: proviene del negocio.
- **Surrogate**: artificial (ej.: ID autoincremental).

---

46. Cargar millones de registros eficientemente

- Bulk load
- Particiones
- Desactivar índices y constraints temporales
- Usar batch inserts

---

47. Temporary table

Tabla temporal de sesión; útil para cálculos intermedios.

---

48. Materialized result cache

Cachea resultados de consultas repetidas para acelerar desempeño.

---

49. Detectar anomalías con solo SQL

- Outliers mediante window functions
- Duplicados
- Valores faltantes
- Comparar medias vs valores actuales

Ejemplo:

```
SELECT *
FROM ventas
WHERE monto > AVG(monto) OVER() * 3;
```

---

50. Consideraciones al escribir SQL para analítica

- Evitar subconsultas innecesarias
- Preagregar datos
- Particionar por fechas
- Usar window functions
- Esquema tipo estrella