Contents

1 Subqueries

1.1 Set membership (`IN`, `NOT IN`):

Se utiliza para verificar si un valor coincide con algún valor dentro de un conjunto de resultados o una lista de valores.

1.1.1 Sintaxis:

```
SELECT column_name
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

1.2 Set comparison (`ALL`, `SOME`):

`ALL` verifica si una condición es verdadera para todos los valores en una subconsulta, mientras que `SOME` verifica si es verdadera para al menos un valor.

1.2.1 Sintaxis:

```
SELECT column_name
FROM table_name
WHERE column_name > ALL (subquery);
```

1.3 Empty relations (`EXISTS`, `NOT EXISTS`):

`EXISTS` verifica si una subconsulta devuelve alguna fila, indicando una relación no vacía. `NOT EXISTS` verifica si hay una relación vacía.

1.3.1 Sintaxis:

```
SELECT column_name
FROM table_name
WHERE EXISTS (subquery);
```

1.4 Subconsulta Correlacionada:

Este tipo de subconsulta depende de la consulta externa para sus valores y se ejecuta repetidamente para cada fila en la consulta externa.

1.4.1 Ejemplo:

```
SELECT column_name
FROM table_name t1
WHERE column_name = (SELECT column_name FROM table_name t2 WHERE t1.column_name = t2.colu
```

1.5 Subconsulta Escalar:

Esta subconsulta devuelve un solo valor y se puede utilizar en varias cláusulas como `SELECT`, `WHERE` y `HAVING`.

1.5.1 Ejemplo:

```
SELECT column_name
FROM table_name
WHERE column_name = (SELECT avg(value) FROM table_name);
```

1.6 CTE (Common Table Expresion):

Un conjunto de resultados temporal nombrado definido dentro de una consulta, mejorando la legibilidad y la modularidad.

1.6.1 Sintaxis:

```
WITH cte_name AS (
    SELECT column_name
    FROM table_name
)
SELECT *
FROM cte_name;
```

1.7 Agregaciones:

Funciones que operan sobre un conjunto de valores para devolver un solo valor, como `AVG`, `MAX`, `MIN`, `COUNT` y `SUM`.

1.7.1 Sintaxis:

```
SELECT AVG(column_name)
FROM table_name;
```

1.8 `GROUP BY`:

Agrupa filas con los mismos valores en columnas especificadas, permitiendo que se apliquen funciones de agregación a cada grupo.

1.8.1 Sintaxis:

```
SELECT column_name, COUNT(*)
FROM table_name
GROUP BY column_name;
```

1.9 `HAVING`:

Filtra los resultados de las agregaciones de `GROUP BY` basándose en condiciones especificadas.

1.9.1 Sintaxis:

```
SELECT column_name, COUNT(*)
FROM table_name
GROUP BY column_name
HAVING COUNT(*) > value;
```

2 Triggers (Sintaxis)

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON table_name FOR EACH ROW
BEGIN
    [trigger_order]
    trigger_body
END;
```

2.1 trigger_name

Este es el nombre que asignas al trigger. Debe ser único dentro de la base de datos para evitar conflictos con otros triggers.

2.2 trigger_time

2.2.1 BEFORE

Cuando se especifica como `BEFORE`, el trigger se ejecutará antes de que ocurra el evento que lo activa.

2.2.2 AFTER

Cuando se especifica como `AFTER`, el trigger se ejecutará después de que haya ocurrido el evento que lo activa.

2.3 trigger_event

2.3.1 INSERT

El trigger se activará cuando se inserte una nueva fila en la tabla especificada.

2.3.2 UPDATE

El trigger se activará cuando se actualice una fila existente en la tabla especificada.

2.3.3 DELETE

El trigger se activará cuando se elimine una fila de la tabla especificada.

2.4 FOR EACH ROW

Esta cláusula indica que el trigger se ejecutará para cada fila afectada por el evento que lo activa.

2.5 trigger_order

2.5.1 FOLLOWS

Cuando se especifica como `FOLLOWS other_trigger_name`, esto indica que el trigger actual debe ejecutarse después del trigger especificado.

2.5.2 PRECEDES

Cuando se especifica como `PRECEDES other_trigger_name`, esto indica que el trigger actual debe ejecutarse antes del trigger especificado.

2.6 trigger_body

El `trigger_body` contiene las declaraciones SQL que definen qué acciones realizará el trigger cuando se active.

3 Transactions

3.1 Que es una transaccion?

Una transacción de base de datos es una unidad lógica de trabajo que agrupa una o más operaciones de base de datos.

3.1.1 Properties

3.1.1.1 Atomicity: La transacción se ejecuta completamente o no se ejecuta en absoluto (el principio de "todo o nada").

3.1.1.2 Consistency: La base de datos pasa de un estado válido a otro, manteniendo las reglas de integridad de los datos.

3.1.1.3 Isolation: Las transacciones se ejecutan de manera independiente, con efectos intermedios ocultos para otros.

3.1.1.4 Durability: Una vez confirmados, los efectos de la transacción son permanentes, persistiendo incluso a través de fallos del sistema.

3.2 What are the different states of a transaction?

3.2.1 Active

La transacción ha comenzado y actualmente está ejecutando operaciones de lectura/escritura.

3.2.2 Committed

Todas las operaciones se han ejecutado, pero los cambios aún no se han guardado permanentemente. Este estado ocurre justo antes del commit.

3.2.3 Failed

Ocurrió un error, impidiendo que la transacción se complete. Esto podría deberse a errores lógicos, fallos del sistema o violaciones de restricciones.

3.2.4 Aborted

La transacción ha sido revertida, deshaciendo todos los cambios y revirtiendo la base de datos a su estado antes de que comenzara la transacción.

3.2.5 Committed

La transacción se completó con éxito y todos los cambios se almacenan permanentemente.

3.3 What are the different types of transactions?

3.3.1 Flat Transactions

La forma más simple, que consiste en una secuencia de operaciones tratadas como una sola unidad.

3.3.2 Nested Transactions

Estas contienen sub-transacciones dentro de una transacción principal, permitiendo un control más detallado. Si una sub-transacción falla, puede revertirse de manera independiente sin afectar a toda la transacción.

3.3.3 Distributed Transactions

Estas implican operaciones que abarcan múltiples bases de datos o sistemas, requiriendo un coordinador para asegurar que todas las partes se completen o se reviertan juntas.

3.4 What are concurrency problems in databases, and how do they relate to transactions?

3.4.1 Dirty Read

Una transacción lee datos que otra transacción aún no ha confirmado.

3.4.2 Non-Repeatable Read

Una transacción lee los mismos datos dos veces y obtiene valores diferentes debido a modificaciones concurrentes.

3.4.3 Phantom Read

Una transacción vuelve a ejecutar una consulta y recupera un conjunto diferente de filas debido a las modificaciones de otra transacción.

3.5 How are concurrency problems addressed in database systems?

3.5.1 Locking

Los bloqueos compartidos (para lectura) y los bloqueos exclusivos (para escritura) previenen operaciones en conflicto sobre los datos. El bloqueo en dos fases (fases de crecimiento y reducción) es una implementación estándar para asegurar la consistencia.

3.5.2 Timestamping

Cada transacción recibe una marca de tiempo única para determinar el orden de ejecución, previniendo anomalías específicas pero potencialmente llevando a más abortos de transacciones.

3.5.3 Multi-Version Concurrency Control (MVCC)

Se mantienen múltiples versiones de los datos, permitiendo que las transacciones accedan a instantáneas consistentes sin bloqueos.

3.6 What are isolation levels, and why are they important?

3.6.1 Read Uncommitted

Permite leer datos no confirmados, susceptible a lecturas sucias.

3.6.2 Read Committed

Previene lecturas sucias asegurando que las transacciones solo lean datos confirmados.

3.6.3 Repeatable Read

Garantiza que leer los mismos datos múltiples veces dentro de una transacción devolverá los mismos valores.

3.6.4 Serializable

Proporciona el aislamiento más fuerte, asegurando que las transacciones se ejecuten como si estuvieran ocurriendo en completo aislamiento.

3.7 Transaction Control Statements in MySQL

3.7.1 SET TRANSACTION

Se utiliza para establecer el nivel de aislamiento para una transacción en MySQL.

```
SET [GLOBAL | SESSION] TRANSACTION transaction_characteristic [, transaction_characterist
```

Ejemplo:

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

3.7.2 COMMIT

Se utiliza para completar con éxito una transacción, haciendo que los cambios sean permanentes.

```
COMMIT;
```

3.7.3 ROLLBACK

Se utiliza para deshacer completamente una transacción, revirtiendo la base de datos a su estado antes de que comenzara la transacción.

```
ROLLBACK;
```

3.7.4 ROLLBACK TO

Se utiliza para deshacer una transacción hasta un punto de guardado específico dentro de una transacción anidada.

```
ROLLBACK TO <savepoint_name>;
```

Ejemplo:

```
ROLLBACK TO my_savepoint;
```

3.7.5 SAVEPOINT

Crea un punto de guardado dentro de una transacción al que puedes volver más tarde.

```
SAVEPOINT <savepoint_name>;
```

Ejemplo:

```
SAVEPOINT my_savepoint;
```

3.7.6 SELECT ... FOR UPDATE

Establece un bloqueo exclusivo en las filas seleccionadas para que otras transacciones no puedan actualizarlas hasta que tu transacción confirme.

```
SELECT * FROM table_name WHERE condition FOR UPDATE;
```

Ejemplo:

```
SELECT * FROM accounts WHERE account_id = 1 FOR UPDATE;
```

3.7.7 SELECT ... FOR SHARE

Establece un bloqueo compartido en las filas seleccionadas para que otras sesiones puedan leerlas pero no modificarlas hasta que tu transacción confirme.

```
SELECT * FROM table_name WHERE condition FOR SHARE;
```

Ejemplo:

```
SELECT * FROM accounts WHERE account_id = 1 FOR SHARE;
```

3.7.8 LOCK TABLES and UNLOCK TABLES

Se utiliza para bloquear y desbloquear tablas para la sesión actual.

```
LOCK TABLES table_name [AS alias] lock_type [, ...];
UNLOCK TABLES;
```

Ejemplo:

```
LOCK TABLES accounts WRITE;
-- Realizar operaciones
UNLOCK TABLES;
```

4 Tips

4.1 JOINS

4.1.1 Cuando usar LEFT o RIGHT join

Cuando pueden haber valores nulos.

Por ejemplo, un pais que no tiene capital.

4.1.2 Muchos joins seguidos

Se pueden concatenar varios joins

```
SELECT
    a.district,
    COUNT(r.rental_id) AS rental_count
FROM rental AS r
INNER JOIN inventory AS i
    ON r.inventory_id = i.inventory_id
INNER JOIN store AS s
    ON i.store_id = s.store_id
INNER JOIN address AS a
    ON s.address_id = a.address_id
GROUP BY a.district;
```

4.2 SELECT

4.2.1 SELECT DISTINCT

se utiliza para eliminar las filas duplicadas

4.3 Preferir agregaciones

Ante la duda, preferir agregaciones antes que ORDER BY y LIMIT.

4.4 Crear tabla

Suele ser buena idea asignarle una ID especifica a cada tabla