MySQL

Digital House - Curso de Desarrollo Web Full Stack

SELECT – Sub-query

Es posible escribir una query para que se ejecute dentro de otra. A la query interna (anidada o “nested”) se la llama **Sub-query**.

Obtener el **máximo rating**

```
SELECT MAX(rating)
FROM pelicula;
```

Obtener las películas con **máximo rating**

```
SELECT p.id, p.titulo, p.rating
FROM pelicula AS p
WHERE
    rating = (
        SELECT MAX(p.rating)
        FROM pelicula p
    )
```

SELECT – Sub-query

Ventajas:

- ✓ Permite **estructurar** las consultas aislando cada parte
- ✓ En algunos casos puede resultar más sencillo usar sub-queries en lugar de joins complejos
- ✓ Facilita la lectura/entendimiento de la query

SELECT – Sub-query

Obtener todos los actores y mostrar en cuántas películas y en cuántos episodios actuaron.

```sql
-- Con sub-queries
SELECT a.id, a.nombre, a.apellido,
    (SELECT COUNT(*) FROM actor_pelicula ap WHERE ap.id_actor = a.id) AS tot_peliculas,
    (SELECT COUNT(*) FROM actor_episodio ae WHERE ae.id_actor = a.id) AS tot_episodios
FROM actor a;

-- Sin sub-queries
SELECT a.id, a.nombre, a.apellido,
    COUNT(ap.id) AS tot_peliculas,
    COUNT(ae.id) AS tot_episodios
FROM actor a
    LEFT JOIN actor_pelicula ap ON ap.id_actor = a.id
    LEFT JOIN actor_episodio ae ON ae.id_actor = a.id
GROUP BY a.id, a.nombre, a.apellido;
```

¿Cómo descartar los actores que no trabajaron en películas?

SELECT – Sub-query

Obtener los actores que hayan actuado en más de 2 películas

```sql
-- Sin sub-queries
SELECT a.id, a.nombre, a.apellido, COUNT(*) AS count_peliculas
FROM actor a
    INNER JOIN actor_pelicula ap ON ap.id_actor = a.id
GROUP BY a.id, a.nombre, a.apellido
HAVING count_peliculas > 2;

-- Con sub-queries
SELECT a.id, a.nombre, a.apellido, count_peliculas.total
FROM actor AS a
    INNER JOIN (
        SELECT ap.id_actor, COUNT(*) AS total
        FROM actor_pelicula AS ap
        GROUP BY ap.id_actor
    ) AS count_peliculas ON count_peliculas.id_actor = a.id
WHERE
    count_peliculas.total > 2;
```

Ejercicio

Obtener aquellas películas que no sean preferidas por ningún actor.

***Importante**: Asegurarse de que existan datos en la BD que cumplan con lo requerido.*

SELECT – LEFT JOIN / IS NULL

Obtener aquellas películas que no sean preferidas por ningún actor

```
SELECT p.id, p.titulo
FROM pelicula AS p
    LEFT JOIN actor AS a ON a.id_pelicula_preferida = p.id
WHERE a.id IS NULL;
```

SELECT – NOT EXISTS / Sub-query

Obtener aquellas películas que no sean preferidas por ningún actor

```
SELECT p.id, p.titulo
FROM pelicula AS p
WHERE NOT EXISTS (
    SELECT a.id
    FROM actor AS a
    WHERE a.id_pelicula_preferida = p.id
);
```

Otro ejercicio

Obtener aquellos actores que hayan actuado tanto en películas como en series.

***Importante**: Asegurarse de que existan datos en la BD que cumplan con lo requerido.*

SELECT

Obtener aquellos actores que hayan actuado tanto en películas como en series.

```
SELECT a.id, a.nombre, a.apellido
FROM actor AS a
    INNER JOIN actor_pelicula ap ON ap.id_actor = a.id
    INNER JOIN actor_episodio ae ON ae.id_actor = a.id
;
```

¿Qué ocurre si un actor actúa en una pelicula y en 2 episodios de una serie?

Utilizar DISTINCT o GROUP BY

SELECT – EXISTS

Obtener aquellos actores que hayan actuado tanto en películas como en series.

Resolver utilizando EXISTS.

```
SELECT a.id, a.nombre, a.apellido
FROM actor AS a
WHERE EXISTS (SELECT ap.id FROM actor_pelicula ap WHERE ap.id_actor = a.id)
  AND EXISTS (SELECT ae.id FROM actor_episodio ae WHERE ae.id_actor = a.id);
```

SELECT – IN / NOT IN

Obtener aquellas películas con 2 ó más actores.

```
SELECT p.id, p.titulo
FROM pelicula AS p
WHERE p.id IN (
    SELECT ap.id_pelicula
    FROM actor_pelicula AS ap
    GROUP BY ap.id_pelicula
    HAVING COUNT(*) >= 2
    );
```

Obtener aquellas películas que no sean preferidas por ningún actor

```
SELECT p.id, p.titulo
FROM pelicula AS p
WHERE p.id NOT IN
    (SELECT a.id_pelicula_preferida
    FROM actor AS a);
```

¿Hay algún problema con esta query?

Práctica

Transacciones en las Bases de Datos

Una **transacción** es una secuencia de operaciones sobre una BD (por lo general modificaciones de datos) que se ejecutan como una única unidad de trabajo.

Por definición deben ser **atómicas**, **consistentes**, **aisladas** y **durables**.

Estas propiedades se las conocen como **ACID** (por sus siglas en inglés **A**tomicity, **C**onsistency, **I**solation, **D**urability)

ACID - Atomicidad

Todas las modificaciones de una transacción deben ser ejecutadas o bien, ninguna de ella. Se ejecuta “todo o nada”.

Ejemplo (simplificado): Pedro transfiere $1000 a la cuenta bancaria de Pablo.

```
UPDATE Saldos SET Saldo = Saldo - 1000 WHERE Cuenta = 'Nro_Cuenta_Pedro';
```


```
UPDATE Saldos SET Saldo = Saldo + 1000 WHERE Cuenta = 'Nro_Cuenta_Pablo';
```

ACID - Consistencia

La transacción comienza a ejecutarse con la BD en un estado válido (consistente) y finaliza estando en otro estado válido.

Para eso, deben cumplirse todas las reglas de **integridad** que hayan sido definidas (e.g.: FOREIGN KEY, UNIQUE constraints).

ACID - Aislamiento

Permite que dos o más transacciones que se ejecuten al mismo tiempo (concurrentes) lo hagan sin afectarse entre sí, como si fueran ejecutadas una detrás de la otra.

Si ambas transacciones deben trabajar sobre el mismo dato una de ellas deberá esperar a que la otra termine.

ACID - Durabilidad

Asegura que una vez realizada la operación va a **persistir** (quedarán almacenados los cambios en la BD) por más que falle el sistema.

Uso del Transaction Log del DBMS.

Ejemplo de una transacción

```
START TRANSACTION;

UPDATE Saldos SET Saldo = Saldo - 1000 WHERE Cuenta = ‘Nro_Cuenta_Pedro’;

-- ...

UPDATE Saldos SET Saldo = Saldo + 1000 WHERE Cuenta = ‘Nro_Cuenta_Pablo’;

COMMIT;
```

COMMIT: Confirma todos los cambios y los guarda en disco.
ROLLBACK: deshace todos los cambios de la transacción (desde el último COMMIT)

Índices - Keys

Se utilizan para obtener más rápidos los registros requeridos en una consulta.

El índice se crea sobre uno o más campos de una tabla y se almacena en una **estructura de datos** cada valor del campo con su posición en la tabla.

Cuando ejecuta una consulta el motor de BD siempre va a intentar utilizar los índices que existan en las tablas involucradas. Caso contrario tiene que rastrear toda la tabla en busca del dato (“full scan”)

Índices

- PRIMARY KEY
- FOREIGN KEY
- UNIQUE
- INDEX (o KEY)

Los índices pueden ser simples o compuestos, dependiendo si se crean sobre 1 o más campos. Por ejemplo: (tipo_doc, nro_doc).

Índices

```
ALTER TABLE pelicula
    ADD KEY `idx-pelicula-rating` (rating);

ALTER TABLE pelicula
    DROP KEY `idx-pelicula-rating`;
```

Índices – Cómo y cuándo se utilizan

- En el WHERE.
- En JOINs. Los campos relacionados deben ser del mismo tipo y tamaño.
- En el ORDER BY.
- Si hay más de un índice posible va a utilizar el que encuentre menos filas.
- Un mismo índice compuesto se utiliza para una parte del mismo.
- Un índice sobre (tipo_doc, nro_doc) se utiliza al buscar (tipo_doc) y también (tipo_doc, nro_doc)
- **Covering Index**: `SELECT nro_doc FROM Personas WHERE tipo_doc = 'DNI';`

EXPLAIN

- Prefijo que se le agrega a una sentencia SELECT para entender cómo es ejecutada por el motor DBMS, qué índices utiliza, cuántos registros trae, etc.

```
EXPLAIN SELECT * FROM Pelicula;
```

EXPLAIN

```
1 • EXPLAIN SELECT * FROM Pelicula;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Pelicula	NULL	ALL	NULL	NULL	NULL	NULL	21	100.00	NULL

```
1 • EXPLAIN SELECT id FROM Pelicula;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

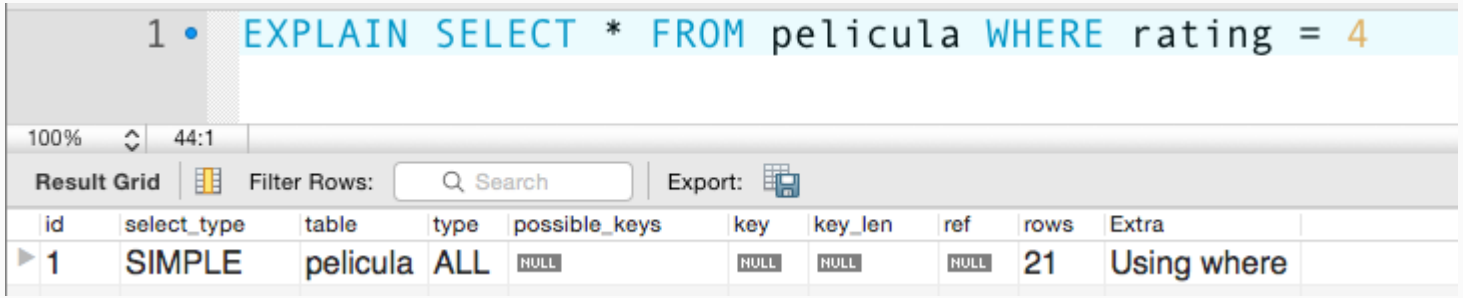
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Pelicula	NULL	index	NULL	id_genero	5	NULL	21	100.00	Using index

```
1 • EXPLAIN SELECT * FROM Pelicula WHERE id = 4;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Pelicula	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL

EXPLAIN - Sin índice por rating



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	pelicula	ALL	NULL	NULL	NULL	NULL	21	Using where

EXPLAIN - Con índice por rating

```
ALTER TABLE pelicula
    ADD KEY `idx-pelicula-rating` (rating);
```

```
1 • EXPLAIN SELECT * FROM pelicula WHERE rating = 4
```

100% 17:1

Result Grid | Filter Rows: Search | Export:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	pelicula	ref	idx-pelicula-rating	idx-pelicula-rating	2	const	1	NULL

EXPLAIN – Para probar …

- Hacer un EXPLAIN SELECT de una query con sub-query. Comparar distintas ejecuciones (con JOINs, con IN, con EXISTS, etc.)

- Crear un índice compuesto de 2 campos, uno de ellos ponerlo en el SELECT y el otro usarlo en el WHERE. Probar hacer el EXPLAIN SELECT antes y después de crear el índice.

Práctica