Práctica 01 - Instalación y Motores de Almacenamiento en MySQL

Pablo Corzo, Pilar Fernandez y José Pascual

1. Análisis de consultas y tiempos (base de datos Sakila)

1.1. SELECT COUNT(*) AS total_movies FROM film

Duraciones: 0.0004415 s y 0.00045725 s
Esta es una consulta simple que recorre todos los registros de la tabla film para contarlos. Es muy ligera, sin operaciones complejas, por eso ambas ejecuciones fueron muy rápidas y estables y tiene una baja duracion.

1.2. SELECT AVG(replacement_cost) AS avg_replacement_cost FROM film

Duraciones: 0.00058275 s y 0.00064 s
Esta consulta es más costosa que la anterior debido al uso de una función de agregación (AVG). Aun así, sigue siendo muy eficiente, ya que opera sobre un solo campo numérico.

1.3. SELECT * FROM film

Duraciones: 0.00593875 s y 0.00276175 s
Esta consulta es más pesada porque muestra todos los campos y todos los registros de la tabla. La diferencia entre ambas ejecuciones se debe al caché o a la carga del sistema. Se suelen evitar estas consultas porque son muy costosas a no ser que se necesite toda la información

1.**4. SELECT * FROM film WHERE length > 120 LIMIT 10**

Duraciones: 0.000266 s y 0.00037425 s
Muy rápida debido al uso del filtro length > 120 y del LIMIT 10, esto reduce la cantidad de datos procesados y devueltos. Por esto es una consulta eficiente, incluso si la tabla tiene muchos registros.

1.5. SELECT * FROM film WHERE rating = 'PG'

Duraciones: 0.00164775 s y 0.001604 s
Esta es una consulta con filtro pero sin LIMIT, lo cual hace que devuelva una gran

cantidad de filas. Aunque es más pesada que la anterior, su rendimiento sigue siendo bueno.

1.6. SELECT film.title AS movie, actor.first_name AS name_actor, actor.last_name AS last_name_actor FROM film JOIN film_category ON film.film_id = film_category.film_id JOIN category ON film_category.category_id = category.category_id JOIN film_actor ON film.film_id = film_actor.film_id JOIN actor ON film_actor.actor_id = actor.actor_id

Duraciones: 0.00113875 s y 0.0013065 s
Esta es la consulta más compleja de todas, con múltiples JOIN entre cinco tablas. A pesar de ello el tiempo de ejecución es bajo, lo que indica que el motor de la base de datos y su diseño están bien optimizados.

1.7. SELECT film_id, title, replacement_cost FROM film WHERE replacement_cost > (SELECT AVG(replacement_cost) FROM film) ORDER BY replacement_cost DESC

Duraciones: 0.00116075 s y 0.0013325 s
Consulta con subconsulta, comparación y ordenamiento. Aunque tiene un poco más de carga lógica, los tiempos siguen siendo bajos. Esto demuestra que se manejan correctamente los índices y que la subconsulta no aumenta mucho el tiempo de ejecucion.

2. Conclusiones generales (base de datos Sakila):

1. Las consultas más rápidas son aquellas que usan funciones simples (COUNT, AVG) o aplican LIMIT a consultas mas complejas.
2. Consultas como SELECT * sin filtros son más costosas, especialmente si la tabla tiene muchos registros y no le aplica un LIMIT.
3. Las consultas con JOINs o subconsultas no aumentaron mucho el tiempo de ejecucion, lo que indica un buen diseño de la base de datos y optimización .

3. Análisis de consultas y tiempos (base de datos World)

3.1. SELECT COUNT(*) AS total_cities FROM city

Duraciones: 0.001032 s y 0.0010235 s
Consulta simple y directa que recorre la tabla city para contar el total de registros. Ambos tiempos de ejecucion son muy similares, rápidos y estables, lo que muestra eficiencia en la operación y el tamaño manejable de la tabla.

3.2. SELECT MAX(Population) AS max_population FROM city

Duraciones: 0.00123125 s y 0.00124525 s
Consulta con función de agregación (MAX), también sobre una sola columna. Aunque es levemente más costosa que el COUNT, sigue siendo rápida y estable, demostrando que las funciones agregadas simples no penalizan el rendimiento sumándole mucho mas tiempo de ejecucion.

3.3. SELECT * FROM city

Duraciones: 0.00280825 s y 0.00332025 s
Consulta con un alto tiempo de ejecución que muestra todos los registros y columnas de la tabla city. El tiempo es más alto en comparación con las consultas anteriores, y la diferencia entre ambas ejecuciones sugiere influencia del uso de caché o condiciones del sistema en tiempo real.

3.4. SELECT * FROM city LIMIT 10

Duraciones: 0.00027875 s y 0.000456 s
Consulta muy rápida gracias al uso de LIMIT, que reduce la cantidad de datos procesados y devueltos. Incluso siendo un SELECT *, el LIMIT garantiza eficiencia y reduce el tiempo de ejecucion.

3.5. SELECT * FROM city WHERE CountryCode = 'JPN'

Duraciones: 0.0007945 s y 0.00079925 s
Consulta con un filtro (WHERE) sin límite. Aunque devuelve más registros que con LIMIT, sigue siendo muy rápida y con tiempos casi iguales entre ejecuciones. Esto muestra un buen rendimiento de los filtros sobre la columna CountryCode.

3.6. SELECT country.Name AS Country, COUNT(city.ID) AS num_big_cities FROM country JOIN city ON country.Code = city.CountryCode WHERE city.Population > 1000000 GROUP BY country.Name HAVING COUNT(city.ID) >= 2 ORDER BY num_big_cities DESC

Duraciones: 0.001784 s y 0.00196825 s
Consulta más compleja, con JOIN, WHERE, GROUP BY, HAVING y ORDER BY. A pesar de la complejidad, los tiempos de ejecución se mantienen bajos. Esto demuestra una buena estructuración de índices y eficiencia en el motor al procesar agregaciones y agrupamientos.

3.7. SELECT country.Name AS Country, country.Continent AS continent, country.Population AS population FROM country WHERE country.Population > (SELECT AVG(country.Population) FROM country WHERE country.Continent = country.Continent) ORDER BY country.Continent, country.Population DESC

Duraciones: 0.00064375 s y 0.000632 s
Consulta con subconsulta y ordenamiento. A pesar de tener una lógica más compleja, el rendimiento es excelente y los tiempos de ejecución son bajos. La subconsulta esta optimizada, porque trabaja dentro de una misma tabla (country), lo cual facilita el acceso a los datos.

4. Conclusiones generales (base de datos World):

1. Las consultas más rápidas fueron las que usaron LIMIT, porque reducen la cantidad de datos procesados.
2. Las funciones agregadas que son simples (COUNT, MAX) mantienen tiempos de ejecución bajos y estables.
3. Las consultas con JOIN, GROUP BY, HAVING y subconsultas complejas tuvieron un rendimiento bueno, lo que demuestra que la base de datos world está bien optimizada.