Explain Plan

Para realizar SQL Tunning a nuestras bases de datos, en la mayoría de los casos necesitaremos paciencia y tiempo para analizar con detenimiento cómo está funcionando todo, así en base a eso, determinar medidas de acción para la optimización.

Ver el plan de ejecución en MySQL de una consulta es bastante sencillo :

```
EXPLAIN SELECT * FROM mi_tabla;
```

La tabla que devuelve la sentencia EXPLAIN lista las tablas en el orden en el que MySQL las leería procesando la consulta (la primera fila que aparece sería la primera tabla que se lee y la última fila sería la última tabla que sería leída). Este orden es importante conocerlo ya que nos indicará el plan de ejecución de la consulta y, por tanto, información relevante para realizar nuestras optimizaciones.

Para ver un plan de ejecución más detallado se puede utilizar la siguiente instrucción :

```
EXPLAIN EXTENDED SELECT * FROM mi_tabla;
```

Si empleamos EXTENDED, podemos también emplear SHOW WARNINGS después de ejecutar la consulta EXPLAIN. La sentencia SHOW WARNINGS nos mostrará la consulta reescrita tras la actuación del optimizador. Para emplear esta sentencia, bastará con poner SHOW WARNINGS después de ejecutar la consulta EXPLAIN EXTENDED. Por ejemplo:

```
EXPLAIN EXTENDED SELECT * FROM mi_tabla;
SHOW WARNINGS;
```

Veamos la salida de realizar un explain con un join :

```
mysql> explain select * from clientes cli inner join cuentas cu on cli.id_cliente = cu.id_cliente;
+----+-------------+-------+------------+------+-----------------------------------------------+-----------------+---------+-------------------------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys                                 | key             | key_len | ref                     | rows | filtered | Extra |
+----+-------------+-------+------------+------+-----------------------------------------------+-----------------+---------+-------------------------+------+----------+-------+
|  1 | SIMPLE      | cli   | NULL       | ALL  | PRIMARY,idx_clientes_id_cliente_descripcion | NULL            | NULL    | NULL                    | 5054 |   100.00 | NULL  |
|  1 | SIMPLE      | cu    | NULL       | ref  | fk_cliente_cuentas                            | fk_cliente_cuentas | 5       | banco.cli.id_cliente |    2 |   100.00 | NULL  |
+----+-------------+-------+------------+------+-----------------------------------------------+-----------------+---------+-------------------------+------+----------+-------+
2 rows in set, 1 warning (0,00 sec)
```

Tambien podemos realizar un explain en formato JSON y da informacion mas detallada de la ejecucion de la consulta :

```
EXPLAIN FORMAT = “JSON” SELECT * FROM mi_tabla;
```

Análisis de resultados

En primer lugar veremos los resultados en general para cualquier base de datos y luego lo aplicaremos a MySQL. Podemos dividir los resultados de un explan en dos grandes grupos Accesos y Join.

Tipos de acceso

TABLE FULL SCAN : Se busca toda la tabla los registros que cumplan con la condición.

INDEX UNIQUE SCAN : El acceso busca una única clave a través de un índice único. Eso significa que siempre la búsqueda devolverá un resultado, en otro caso ya no sería UNIQUE. Un ejemplo claro de este tipo de acceso es el que hacemos cuando buscamos a través de la PK. EN MySQL se indica con el tipo “const”.

INDEX MERGE : Para obtener la tabla se debe acceder a dos índices y luego hacer merge de los resultados.

INDEX RANGE SCAN :Como su nombre indica, en este caso no buscamos por un único valor, sino por un rango de ellos, por lo que nos devolverá más de un registro. Esto ocurre cuando usamos los operadores de rango (<, >, <>, <=, >=, BETWEEN). En MySQL se indica con el tipo “range”.

INDEX FULL SCAN :Esta es la peor de las decisiones con índices y para que ocurra deberemos de no filtrar por el índice y que sea el motor el que decida que este acceso es más óptimo que un FULL SCAN, lo cual es raro, ya que, como os comenté en su momento, recorrer un índice es costoso y mejora la optimización sólo si está ordenado y se recorre un rango "pequeño" de registros de la tabla. Ahora bien, si por ejemplo queremos tirar un ORDER BY de una tabla sin filtrado y este orden es el utilizado por el índice, en ese caso entonces el motor decidirá acceder por esta vía.

INDEX NON UNIQUE KEY LOOKUP: El acceso busca una única clave a través de un índice no único.. EN MySQL se indica con el tipo “ref”.

Estas pautas de accesos son a nivel de lectura de los registros de la tabla. También hay otra parte muy importante que debemos tener en cuenta: Como se realizan los JOINs.

Tipos de Join

NESTED LOOP : Este es el JOIN típico. Se recorren los registros de A, y luego se prueba que cada uno de los registros de A esté en B. Es por eso que en otros capítulos dábamos las pautas de que hay que intentar siempre filtrar al máximo la tabla de unión (en el caso exacto la A) y que esta tabla fuente tiene que ser, si es posible, la más pequeña de ambas (incluyendo filtrados).

HASH MERGE JOIN : Esto no está en todos los motores, así que no entraremos demasiado en el tema. Simplemente basta saber que este caso es incluso más eficiente que el Nested Loop, ya que la utilización de un Hash Map para la recuperación de datos es mucho más eficiente.

SORT MERGE JOIN : Este es el peor JOIN que podemos encontrar y tiene lugar cuando hay que ordenar previamente el resultado de ambas tablas, lo que genera mayor carga de datos.

PRODUCTO CARTESIANO: Recorre ambas tablas por completo. Esto ocurre porque la consulta está mal escrita y falta el JOIN entre ambas tablas.

MYSQL EXPLAIN

- **id:** muestra el número secuencial que identifica cada una de las tablas que se procesan en la consulta realizada.
- **select_type:** muestra el tipo de SELECT que se ha ejecutado. En función del tipo de SELECT nos permitirá identificar qué tipo de consulta se trata. Existen varios tipos distintos: SIMPLE, PRIMARY, UNION.
- **table:** muestra el nombre de la tabla a la que se refiere la información de la fila. También, puede ser alguno de los siguientes valores:
    - tabla resultante de la unión de las tablas cuyos campos id son M y N.
    - tabla resultante de una tabla derivada cuyo id es N. Una tabla derivada puede ser, por ejemplo, una subconsulta en la cláusula FROM.
    - tabla resultante de una subconsulta materializada cuyo id es N.
- **type:** muestra el tipo de unión que se ha empleado, ver al final para mas informacion
  possible_keys: muestra qué índices puede escoger MySQL para buscar las filas de la tabla. Si esta columna es NULL, significa que no hay índices que puedan ser usados para mejorar la consulta. En este caso, podría ser interesante examinar las columnas empleadas en el WHERE para analizar si hay alguna columna que pueda emplearse para construir un índice.
- **key:** muestra el índice que MySQL ha decido usar para ejecutar la consulta. Es posible que el nombre del índice no esté presente en la lista de possible_keys.
- **key_len:** muestra el tamaño del índice que MySQL ha decidido usar para ejecutar la consulta.
- **ref:** muestra qué columnas o constantes son comparadas con el índices especificado en la columna key.
- **rows:** muestra el número de filas que MySQL cree que deben ser examinadas para ejecutar la consulta. Este número es un número aproximado.
- **filtered:** muestra el porcentaje estimado de filas que serán filtradas por la condición de la consulta. Esta columna sólo se muestra cuando se emplea EXPLAIN EXTENDED.
- **Extra:** muestra información adicional sobre cómo MySQL ejecuta la consulta. Esto podría indicar ordenamiento o agrupamiento.

El campo más importante de este set de datos es el campo type, el cual nos va a indicar el tipo de acceso que se está teniendo a la relación especificado :

Nombre de sistema	Color	Texto en el diagrama visual	Descripción.
SYSTEM	Azul	Single row: system constant	Muy bajo costo
CONST	Azul	Single row: constant	Muy bajo costo
EQ_REF	Verde	Unique Key Lookup	Bajo costo , el optimizador pudo encontrar un índice que le permite obtener los registros.
REF	Verde	Non-Unique Key Lookup	Bajo costo - Medio. Bajo si el número de filas que cumplemn la condición es bajo , pero se incrementa mientras crecen los números de registros
FULLTEXT	Amarillo	Fulltext Index Search	Costo bajo .Especializado en FULLTEXT índices.
REF_OR_NULL	Verde	Key Lookup + Fetch NULL Values	Realiza una busqueda incluyendo los valores nulos (REF). Costo medio - bajoLow-medium
INDEX_MERGE	Verde	Index Merge	Medio - Realiza un merge sobre dos índices con el objetivo de obtener los datos.Es conveniente buscar una mejor estrategia de índices. En MySQL hay tres tipos : **MERGE INTERSECTION:** Realiza escaneos simultáneos en todos los índices usados y devuelve la intersección de lo mismos. **MERGE UNION :** No se devuelve la interseccion sino la union de ambos índices. **MERGE SORT-UNION :** Similar al anterior pero debe buscar los RowId y luego ordenar el resultado.

UNIQUE_SUBQUERY	Naranja	Unique Key Lookup into table of subquery	Bajo , utilizado para optimizar subqueries
INDEX_SUBQUERY	Naranja	Non-Unique Key Lookup into table of subquery	Bajo , utilizado para optimizar subqueries
RANGE	Amarillo	Index Range Scan	Medio, busqueda parcial dentro de un indice.
INDEX	Rojo	Full Index Scan	Alto , especialmente en índices grandes-
ALL	Rojo	Full Table Scan	Muy alto, se hace un full scan de toda la tabla