Introducción

En este laboratorio, vamos a explorar diferentes técnicas para analizar y mejorar el rendimiento de consultas en MySQL, principalmente consultas que duren varios segundos.

Objetivos

Los objetivos de este laboratorio son:

Aprender a añadir y eliminar un índice en una tabla.

Aprender a añadir y eliminar un histograma en una tabla.

Utilizar el comando Explain para analizar el rendimiento de una consulta.

Utilizar el esquema de rendimiento para obtener métricas de rendimiento de una consulta.

Requisitos previos

Para realizar este laboratorio utilizaremos el servidor de bases de datos que obtuvimos como resultado del Laboratorio 1.

Procedimiento a seguir

A continuación, se describen los pasos a realizar, suponiendo que el servidor de bases de datos y la conexión de red están correctamente configurados.

1 Preparación

A) Configuración de MySQL

Antes de comenzar, es conveniente revisar que los *logs* utilizados en los laboratorios 3 y 4 están desactivados (excepto el log de errores). Para ello, revisar el fichero de configuración de MySQL:

```
  GNU nano 4.8              /etc/mysql/mysql.conf.d/mysqld.cnf              Modified
# * Logging and Replication
#
# Both location gets rotated by the cronjob.
#
# Log all queries
# Be aware that this log type is a performance killer.
#general_log_file        = /var/log/mysql/query.log
#general_log             = 1
#
# Error log - should be very few entries.
#
log_error = /var/log/mysql/error.log
#
# Here you can see queries with especially long duration
#slow_query_log         = 1
#slow_query_log_file    = /var/log/mysql/mysql-slow.log
#long_query_time = 2
# log-queries-not-using-indexes

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell ^_ Go To Line
```

Después, configurar el modo de aislamiento al nivel “Repetable Read” (ver Laboratorio 5):

```
mysql> select @@TRANSACTION_ISOLATION;
+-------------------------+
| @@TRANSACTION_ISOLATION |
+-------------------------+
| REPEATABLE-READ         |
+-------------------------+
1 row in set (0.00 sec)
```

B) Instalación de la base de datos Employees

En este laboratorio utilizaremos *Employees*, una base de datos de muestra mantenida por MySQL que contiene los datos ficticios sobre los empleados de una empresa. Toda la información de la misma se puede consultar en: https://dev.mysql.com/doc/employee/en/

Los ficheros para instalar la base de datos se encuentran en el siguiente repositorio de GitHub: https://github.com/datacharmer/test_db

Para instalar *Employees*, descargar todos los ficheros del repositorio en formato comprimido ZIP en el servidor de MySQL y descomprimirlos.

```
ubuntu@ip-172-31-18-109:~$ wget https://github.com/datacharmer/test_db/archive/refs/heads/master.zip
--2021-05-02 07:48:50--  https://github.com/datacharmer/test_db/archive/refs/heads/master.zip
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/datacharmer/test_db/zip/refs/heads/master [following]
--2021-05-02 07:48:51--  https://codeload.github.com/datacharmer/test_db/zip/refs/heads/master
Resolving codeload.github.com (codeload.github.com)... 140.82.113.10
Connecting to codeload.github.com (codeload.github.com)|140.82.113.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: 'master.zip'

master.zip                  [       <=>                         ]  34.99M  3.28MB/s    in 11s

2021-05-02 07:49:01 (3.28 MB/s) - 'master.zip' saved [36688498]

ubuntu@ip-172-31-18-109:~$ unzip master.zip
Archive:  master.zip
e5f310ac7786a2a181a7fc124973725d7aa4ce7c
   creating: test_db-master/
  inflating: test_db-master/Changelog

...
  inflating: test_db-master/test_employees_sha.sql
  inflating: test_db-master/test_versions.sh
ubuntu@ip-172-31-18-109:~$
```

Una vez descomprimidos, utilizar el siguiente comando para instalar la base de datos desde la carpeta recién creada (puede tardar hasta 1 minuto):

```
:~$ cd test_db-master
:~/test_db-master$ mysql -u root -p < employees.sql
```

```
ubuntu@ip-172-31-18-109:~$ cd test_db-master/
ubuntu@ip-172-31-18-109:~/test_db-master$ mysql -u root -p < employees.sql
Enter password:
INFO
CREATING DATABASE STRUCTURE
INFO
storage engine: InnoDB
INFO
LOADING departments
INFO
LOADING employees
INFO
LOADING dept_emp
INFO
LOADING dept_manager
INFO
LOADING titles
INFO
LOADING salaries
data_load_time_diff
00:01:05
```

Finalmente, entrar a la consola de MySQL y verificar que la base de datos se ha cargado correctamente.

```
mysql> use employees;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------------+
| Tables_in_employees  |
+----------------------+
| current_dept_emp     |
| departments          |
| dept_emp             |
| dept_emp_latest_date |
| dept_manager         |
| employees            |
| salaries             |
| titles               |
+----------------------+
8 rows in set (0.01 sec)

mysql>
```

2 Técnicas de optimización

En esta sección veremos 2 técnicas que se pueden emplear para reducir el tiempo de ejecución de consultas en MySQL. Para ello, utilizaremos la tabla *salaries* de la base de datos, que contiene los sueldos de los empleados listados en *Employees*.

A) Índices

Un índice es una serie de meta-información que un SGBD utiliza para reducir los tiempos en operaciones de tipo SELECT. Los índices actúan como punteros a las diferentes filas de una tabla para una (o varias) columnas, los cuales permiten determinar de una forma más óptima qué filas coinciden con la cláusula WHERE de una consulta.

En MySQL se pueden utilizar diferentes tipos de índices:

- *B-Tree*: es el índice que se crea por defecto, ya que se adapta a la mayoría de tipos de datos. Se crea un árbol de tipo B-Tree con los valores de la columna indexada.
- *Hash*: en lugar de un árbol, se crean valores *hash* para las entradas de la columna. Sólo es válido para condiciones = o <=>, pero puede superar en rendimiento a los B-Tree en ellas. Más información: https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html
- *Full Text*: orientado a campos varchar/text que contengan varias frases o al menos varias palabras. Más información: https://dev.mysql.com/doc/refman/8.0/en/fulltext-search.html
- *Spatial*: utiliza árboles R-Tree en lugar de B-Tree. Se utiliza para datos geográficos, generalmente en formato longitud/latitud. Más información: https://dev.mysql.com/doc/refman/8.0/en/spatial-index-optimization.html
- *Multi-value*: se utiliza para campos que almacenen datos en formato JSON dentro de una tabla.

En este laboratorio sólo trabajaremos el primero de ellos, pero toda la información sobre índices en MySQL se puede obtener en: https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html

Utilizaremos como ejemplo la siguiente consulta, que obtiene el número de empleados que tienen (o han tenido) un sueldo inferior a 40000 dólares de la tabla *salaries*:

```
mysql> select * from salaries where salary < 40000;
```

Ya que nos interesa verificar la efectividad de los índices, ejecutamos la consulta varias veces para saber cuanto tarda sin hacer ninguna modificación a la tabla:

```
mysql> select count(*) from salaries where salary < 40000;
+----------+
| count(*) |
+----------+
|    11711 |
+----------+
1 row in set (0.56 sec)

mysql> select count(*) from salaries where salary < 40000;
+----------+
| count(*) |
+----------+
|    11711 |
+----------+
1 row in set (0.49 sec)

mysql> select count(*) from salaries where salary < 40000;
+----------+
| count(*) |
+----------+
|    11711 |
+----------+
1 row in set (0.49 sec)
```

El mejor tiempo que se obtiene es 0.49 segundos.

NOTA: Los valores de tiempos en vuestro servidor no tienen por qué coincidir con los mostrados aquí, aunque deberían ser similares.

Para intentar mejorar el tiempo de ejecución de esta consulta, añadimos un índice llamado *idx_salary* a la tabla *salaries* en la columna *salary* (puede tardar varios segundos):

```
mysql> alter table salaries add index idx_salary (salary);
```

```
mysql> alter table salaries add index idx_salary (salary);
Query OK, 0 rows affected (12.27 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show create table salaries;
+----------+-----------------------------------------------------------------------
------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+
| Table    | Create Table

                                                                                  |
+----------+-----------------------------------------------------------------------
------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+
| salaries | CREATE TABLE `salaries` (
  `emp_no` int NOT NULL,
  `salary` int NOT NULL,
  `from_date` date NOT NULL,
  `to_date` date NOT NULL,
  PRIMARY KEY (`emp_no`,`from_date`),
  KEY `idx_salary` (`salary`),
  CONSTRAINT `salaries_ibfk_1` FOREIGN KEY (`emp_no`) REFERENCES `employees` (`emp_no`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+----------+-----------------------------------------------------------------------
------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Una vez creado, es recomendable verificar que se ha añadido correctamente usando el comando *Show Create Table* (ver fila KEY ‘idx_salary’).

Tras añadir el índice podemos repetir la consulta anterior y comprobar que el tiempo de ejecución se reduce drásticamente:

```
mysql> select count(*) from salaries where salary < 40000;
+----------+
| count(*) |
+----------+
|    11711 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from salaries where salary < 40000;
+----------+
| count(*) |
+----------+
|    11711 |
+----------+
1 row in set (0.00 sec)
```

Para borrar el índice podemos utilizar el siguiente comando:

```
mysql> alter table salaries drop index idx_salary;
```

```
mysql> alter table salaries drop index idx_salary;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Si después de borrar el índice repetimos el comando anterior, podemos comprobar cómo los tiempos de ejecución vuelven a los números iniciales:

```
mysql> select count(*) from salaries where salary < 40000;
+----------+
| count(*) |
+----------+
|    11711 |
+----------+
1 row in set (1.14 sec)

mysql> select count(*) from salaries where salary < 40000;
+----------+
| count(*) |
+----------+
|    11711 |
+----------+
1 row in set (0.48 sec)

mysql>
```

B) Histogramas

Los histogramas son una nueva característica implementada en MySQL 8, complementaria a los índices. Son un tipo de meta-información que se añade a una columna para mejorar los tiempos de consultas SELECT, pero en lugar de ser una estructura árbol o una tabla Hash, un histograma organiza los datos en grupos (llamados *buckets*) que representan la frecuencia de aparición de cada valor.

El siguiente diagrama[1] representa un ejemplo de histograma sobre una columna con valores de tipo *integer*, donde el valor 7 es el que más se repite en la columna, seguido del 3:

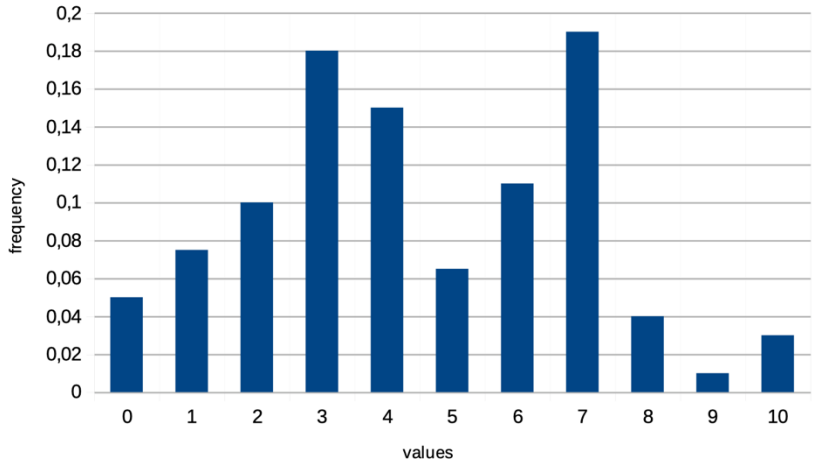

Ilustración 1: Ejemplo de histograma (obtenido de percona.com)

Esta información es utilizada por el optimizador de MySQL para generar el mejor plan posible para ejecutar la consulta. Los histogramas son generalmente apropiados para consultas SELECT que implican la unión de varias tablas y que además requieren ordenar resultados.

Se puede obtener más información sobre los histogramas en la documentación de MySQL:
https://dev.mysql.com/doc/refman/8.0/en/analyze-table.html

[1] Obtenido de: https://www.percona.com/blog/2019/10/29/column-histograms-on-percona-server-and-mysql-8-0/

En esta sección utilizaremos la siguiente consulta a modo de ejemplo:

```
mysql> select count(*) from salaries s join titles t
       on s.emp_no=t.emp_no
       where salary > 90000
       order by salary;
```

Esta consulta devuelve, para cada empleado, la relación de puestos de trabajo que ha tenido (ingeniero, ingeniero jefe, …) junto con los contratos que ha tenido en cada puesto, siempre que el salario haya sido superior a 90.000 dólares, ordenados por nómina.

Se ha añadido al comienzo de la consulta la cláusula *count* para obtener el número de combinaciones totales resultantes de la consulta, en lugar de todos los datos.

```
mysql> select count(*) from salaries s join titles t
    -> on s.emp_no=t.emp_no
    -> where salary > 90000
    -> order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (2.13 sec)

mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary
 > 90000 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (1.94 sec)

mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary
 > 90000 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (1.92 sec)
```

Se ha repetido la consulta 3 veces y el mejor tiempo ha sido de 1.92 segundos. Sin utilizar ninguna técnica de optimización, la consulta puede considerarse lenta.

Ya que la columna *salary* se utiliza para discriminar valores en la consulta, podemos añadir un histograma a ella para optimizar su ejecución. Con el siguiente comando se añade un histograma de 1024 *buckets* (el máximo posible) a la columna *salary*:

```
mysql> analyze table salaries update histogram on salary
                    with 1024 buckets;
```

```
mysql> analyze table salaries update histogram on salary with 1024 buckets;
+--------------------+-----------+----------+-------------------------------------------------+
| Table              | Op        | Msg_type | Msg_text                                        |
+--------------------+-----------+----------+-------------------------------------------------+
| employees.salaries | histogram | status   | Histogram statistics created for column 'salary'. |
+--------------------+-----------+----------+-------------------------------------------------+
1 row in set (3.21 sec)
```

Si repetimos la consulta, el tiempo de ejecución se reduce significativamente:

```
mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000
 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (1.46 sec)

mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000
 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (1.10 sec)

mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000
 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (1.09 sec)
```

Finalmente, para eliminar el histograma recién creado se puede utilizar el siguiente comando:

```
mysql> analyze table salaries drop histogram on salary;
```

```
mysql> analyze table salaries drop histogram on salary;
+--------------------+-----------+----------+-------------------------------------------------+
| Table              | Op        | Msg_type | Msg_text                                        |
+--------------------+-----------+----------+-------------------------------------------------+
| employees.salaries | histogram | status   | Histogram statistics removed for column 'salary'. |
+--------------------+-----------+----------+-------------------------------------------------+
1 row in set (0.01 sec)
```

Si ejecutamos la sentencia de nuevo sin índice ni histograma, su tiempo de ejecución vuelve ser similar al inicial:

```
mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000
 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (2.08 sec)

mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000
 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (1.93 sec)

mysql>
mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000
 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (1.95 sec)
```

3 Análisis del rendimiento

En esta sección se presentan 2 herramientas que permiten obtener más información sobre la ejecución de consultas y sus costes asociados: el comando Explain y el esquema de rendimiento (Performance Schema).

A) Comando Explain

El comando Explain proporciona información sobre cómo MySQL ejecuta una sentencia particular. Es compatible con las sentencias SELECT, DELETE, INSERT, REPLACE y UPDATE. Toda la información sobre Explain se puede consultar en: https://dev.mysql.com/doc/refman/8.0/en/explain-output.html

Para comenzar con esta parte, usaremos una consulta simple que permita entender el funcionamiento de Explain. El siguiente comando devuelve los identificadores de los empleados que han sido (o son) jefes de departamento y durante qué fechas:

```
mysql> select * from dept_manager;
```

```
mysql> select * from dept_manager;
+--------+---------+------------+------------+
| emp_no | dept_no | from_date  | to_date    |
+--------+---------+------------+------------+
| 110022 | d001    | 1985-01-01 | 1991-10-01 |
| 110039 | d001    | 1991-10-01 | 9999-01-01 |
| 110085 | d002    | 1985-01-01 | 1989-12-17 |
| 110114 | d002    | 1989-12-17 | 9999-01-01 |
| 110183 | d003    | 1985-01-01 | 1992-03-21 |
| 110228 | d003    | 1992-03-21 | 9999-01-01 |
| 110303 | d004    | 1985-01-01 | 1988-09-09 |
| 110344 | d004    | 1988-09-09 | 1992-08-02 |
| 110386 | d004    | 1992-08-02 | 1996-08-30 |
| 110420 | d004    | 1996-08-30 | 9999-01-01 |
| 110511 | d005    | 1985-01-01 | 1992-04-25 |
| 110567 | d005    | 1992-04-25 | 9999-01-01 |
| 110725 | d006    | 1985-01-01 | 1989-05-06 |
| 110765 | d006    | 1989-05-06 | 1991-09-12 |
| 110800 | d006    | 1991-09-12 | 1994-06-28 |
| 110854 | d006    | 1994-06-28 | 9999-01-01 |
| 111035 | d007    | 1985-01-01 | 1991-03-07 |
| 111133 | d007    | 1991-03-07 | 9999-01-01 |
| 111400 | d008    | 1985-01-01 | 1991-04-08 |
| 111534 | d008    | 1991-04-08 | 9999-01-01 |
| 111692 | d009    | 1985-01-01 | 1988-10-17 |
| 111784 | d009    | 1988-10-17 | 1992-09-08 |
| 111877 | d009    | 1992-09-08 | 1996-01-03 |
| 111939 | d009    | 1996-01-03 | 9999-01-01 |
+--------+---------+------------+------------+
24 rows in set (0.01 sec)
```

Para obtener información sobre la forma en que MySQL ejecuta un comando, hay que añadir "explain" delante del mismo. Con el comando anterior, quedaría así:

```
mysql> explain select * from dept_manager;
```

```
mysql> explain select * from dept_manager;
+----+-------------+--------------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table        | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+--------------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | dept_manager | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   24 |   100.00 | NULL  |
+----+-------------+--------------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

Esta es la forma básica de uso de Explain, la cual devuelve diferentes valores sobre la sentencia. De ellos, los más relevantes son:

- *select_type*: Tipo de operación SELECT.
- *posible_keys*: Indica los índices que MySQL puede utilizar para ejecutar la consulta.
- *rows*: Número de filas que MySQL estima que recorrerá para obtener el resultado de la consulta.
- *extra*: Información adicional, p.e., si ha usado un índice o una tabla temporal. Todos los valores posibles están en: https://dev.mysql.com/doc/refman/8.0/en/explain-output.html#explain-extra-information

Es importante notar que estos valores provienen de un análisis estático de la consulta, sin que se haya ejecutado. Esto permite obtener información de forma más rápida, pero implica que algunos valores son estimados y no reales.

Sin embargo, la anterior no es la única forma de utilizar Explain. Se puede obtener información más concreta sobre los costes de ejecución y la planificación de la consulta de la siguiente forma:

```
mysql> explain format=tree select * from dept_manager;
```

```
mysql> explain format=tree select * from dept_manager;
+----------------------------------------------------+
| EXPLAIN                                            |
+----------------------------------------------------+
| -> Table scan on dept_manager  (cost=2.65 rows=24)
 |
+----------------------------------------------------+
1 row in set (0.00 sec)
```

Esta forma de uso de Explain muestra las operaciones que MySQL necesita ejecutar para calcular el resultado de la sentencia, junto con el coste de cada paso y las filas a recuperar. El valor unitario del coste es un parámetro interno que MySQL estima en base al tiempo que necesita para leer y escribir una fila en disco.

En el caso concreto de esta sentencia, la única operación es leer la tabla completa, la cual se representa como una ejecución “scan”. Aunque en este caso es necesario, debemos evitar operaciones “scan” completas en las tablas a menos que sean imprescindibles, ya que son las que más deterioran el rendimiento de una sentencia.

Como nota final, esta forma de usar Explain, al igual que la anterior, no ejecuta la sentencia y obtiene la información en base a estimaciones.

Además de las anteriores, hay una tercera forma de utilizar Explain para que proporciona información sobre una ejecución real de una sentencia:

```
mysql> explain analyze select * from dept_manager;
```

```
mysql> explain analyze select * from dept_manager;
+-----------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                             |
+-----------------------------------------------------------------------------------------------------+
| -> Table scan on dept_manager  (cost=2.65 rows=24) (actual time=0.033..0.052 rows=24 loops=1)
 |
+-----------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

En esta forma de uso, la información proporcionada tiene una estructura similar a la vista en la forma anterior pero con una diferencia importante: se proporcionan los tiempos reales que requieren cada paso, en milisegundos.

En la imagen de ejemplo se proporcionan 2 valores de tiempo para el paso “scan”: el primero (0.033 ms) indica cuánto ha tardado en hacer “scan” sobre la primera fila de la tabla y el segundo (0.052 ms) cuánto ha tardado en hacer “scan” sobre todas las filas de la tabla.

Aunque la primera instancia podría pensarse que el segundo valor debería ser una multiplicación del 1º por el número de filas, no tiene por qué ser así. Tanto MySQL como los sistemas operativos actuales tienen técnicas para optimizar lecturas consecutivas que permiten obtener varios valores de filas diferentes en una única lectura.

Para mostrar un ejemplo más completo de Explain, la siguiente imagen muestra el resultado de analizar la consulta utilizada en el apartado anterior de este laboratorio. En este primer análisis, la columna *salary* no tiene ningún índice ni histograma:

```
mysql> explain analyze select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000 order by salary;
+------------------------------------------------------------------------------------------------------------------------------------
```

…

```
| -> Aggregate: count(0)  (actual time=11242.826..11242.827 rows=1 loops=1)
    -> Nested loop inner join  (cost=577526.66 rows=1404339) (actual time=0.256..10992.710 rows=405478 loops=1)
        -> Index scan on t using PRIMARY  (cost=44567.50 rows=442545) (actual time=0.048..363.276 rows=443308 loops=1)
        -> Filter: (s.salary > 90000)  (cost=0.25 rows=3) (actual time=0.019..0.021 rows=1 loops=443308)
            -> Index lookup on s using PRIMARY (emp_no=t.emp_no)  (cost=0.25 rows=10) (actual time=0.004..0.012 rows=10 loops=443308)
 |
```

El análisis muestra el árbol de operaciones, donde la operación principal es la unión la tabla *salaries* (aparece como *s*) filtrada con la tabla *titles* (aparece como *t*). Podemos ver cómo se usan índices para la unión con la clave primaria en el campo *emp_no*. A modo de comparación, la siguiente imagen muestra el mismo análisis tras haber habilitado un histograma en la columna *salary*:

```
mysql> analyze table salaries update histogram on salary with 1024 buckets;
+--------------------+-----------+----------+---------------------------------------------------+
| Table              | Op        | Msg_type | Msg_text                                          |
+--------------------+-----------+----------+---------------------------------------------------+
| employees.salaries | histogram | status   | Histogram statistics created for column 'salary'. |
+--------------------+-----------+----------+---------------------------------------------------+
1 row in set (3.36 sec)
```

```
mysql> explain analyze select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000 order by salary;
+------------------------------------------------------------------------------------------------------------------------------------
```

…

```
------------+
| -> Aggregate: count(0)  (actual time=6394.480..6394.481 rows=1 loops=1)
    -> Nested loop inner join  (cost=378479.78 rows=341075) (actual time=0.137..6142.522 rows=405478 loops=1)
        -> Filter: (s.salary > 90000)  (cost=285372.60 rows=230468) (actual time=0.115..4236.058 rows=234267 loops=1)
            -> Table scan on s  (cost=285372.60 rows=2838426) (actual time=0.040..2319.470 rows=2844047 loops=1)
        -> Index lookup on t using PRIMARY (emp_no=s.emp_no)  (cost=0.26 rows=1) (actual time=0.003..0.004 rows=2 loops=234267)
 |
+------------------------------------------------------------------------------------------------------------------------------------
```

Si comparamos ambos casos podemos ver cómo, aunque la operación principal sigue siendo una unión, en este segundo caso se ha intercambiado el orden de las operaciones (primero se trata la tabla *salaries*). Además, el coste dedicado a procesar la unión se ha reducido de 577.526,66 en la versión inicial a 378.489,78 en la versión con histograma.

B) Performance Schema

Otra forma de obtener información sobre la ejecución de una consulta es el esquema de rendimiento (o Performance Schema). Es un sistema de monitorización integrado en MySQL que recoge métricas de tiempo sobre las diferentes sentencias que se ejecutan en el sistema. Si queremos mejorar los tiempos de ejecución de una consulta, este esquema puede proporcionar información (complementaria a Explain) para detectar qué partes de la consulta son mejorables.

Por defecto, el esquema de rendimiento está activado en MySQL 8. Para verificarlo, utilizar la siguiente consulta:

```
mysql> SHOW VARIABLES LIKE 'performance_schema';
```

```
mysql> SHOW VARIABLES LIKE 'performance_schema';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| performance_schema | ON    |
+--------------------+-------+
1 row in set (0.01 sec)
```

En caso de que el resultado de la consulta fuese “OFF”, cambiar el valor de la variable “performance_schema” del fichero de configuración a “ON” y reiniciar el servicio de MySQL.

En esta sección se presenta un uso básico del esquema de rendimiento, se puede encontrar más información en los siguientes enlaces de la documentación de MySQL:

- https://dev.mysql.com/doc/mysql-perfschema-excerpt/8.0/en/performance-schema-quick-start.html
- https://dev.mysql.com/doc/mysql-perfschema-excerpt/8.0/en/performance-schema-query-profiling.html

Para que el esquema de rendimiento capture todas las métricas de rendimiento posibles, hay que activarlas con las siguientes sentencias:

```
mysql> UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES' WHERE NAME LIKE '%stage/%';

mysql> UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES' WHERE NAME LIKE '%events_statements_%';

mysql> UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES' WHERE NAME LIKE '%events_stages_%';
```

```
mysql> UPDATE performance_schema.setup_instruments
    ->         SET ENABLED = 'YES', TIMED = 'YES'
    ->         WHERE NAME LIKE '%stage/%';
Query OK, 108 rows affected (0.00 sec)
Rows matched: 124  Changed: 108  Warnings: 0
```

```
mysql> UPDATE performance_schema.setup_consumers
    ->         SET ENABLED = 'YES'
    ->         WHERE NAME LIKE '%events_statements_%';
Query OK, 1 row affected (0.00 sec)
Rows matched: 3  Changed: 1  Warnings: 0

mysql> UPDATE performance_schema.setup_consumers
    ->         SET ENABLED = 'YES'
    ->         WHERE NAME LIKE '%events_stages_%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

Una vez ejecutadas, se empiezan a registrar métricas de rendimiento para las consultas que se ejecutan en MySQL.

Utilizaremos de nuevo la consulta que obtiene los datos de los empleados con nómina mayor de 90000 dólares como caso de uso. Para analizar su rendimiento con el Performance Schema, primero, se debe ejecutar de la forma habitual (sin añadir ningún prefijo o sufijo a la sentencia). En el ejemplo se ha eliminado antes el histograma de la columna *salary*.

```
mysql> analyze table salaries drop histogram on salary;
+--------------------+-----------+----------+-----------------------------------------------------+
| Table              | Op        | Msg_type | Msg_text                                            |
+--------------------+-----------+----------+-----------------------------------------------------+
| employees.salaries | histogram | status   | Histogram statistics removed for column 'salary'    |
```

```
mysql> select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000 order by salary;
+----------+
| count(*) |
+----------+
|   405478 |
+----------+
1 row in set (2.48 sec)
```

Una vez que se haya ejecutado, hay que buscar su identificador en el registro del esquema de rendimiento. Para ello utilizar la siguiente consulta, incluyendo algún fragmento representativo de la sentencia a monitorizar en la cláusula “like”. En el ejemplo se utiliza el fragmento “90000”:

```
mysql> SELECT EVENT_ID, TRUNCATE(TIMER_WAIT/1000000000000,6)
                    as Duration, SQL_TEXT
   FROM performance_schema.events_statements_history_long
            WHERE SQL_TEXT like '%<FRAGMENTO>%';
```

```
mysql> SELECT EVENT_ID, TRUNCATE(TIMER_WAIT/1000000000000,6)
    -> as Duration, SQL_TEXT
    -> FROM performance_schema.events_statements_history_long
    -> WHERE SQL_TEXT like '%90000%';
+----------+----------+-----------------------------------------------------------------------------------------------------------------------+
| EVENT_ID | Duration | SQL_TEXT                                                                                                              |
+----------+----------+-----------------------------------------------------------------------------------------------------------------------+
|       26 |   2.4782 | select count(*) from salaries s join titles t on s.emp_no=t.emp_no where salary > 90000 order by salary |
+----------+----------+-----------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

En la consulta de ejemplo, se obtiene que el identificador de la ejecución es 26.

Una vez obtenido el identificador, para leer las métricas del esquema de rendimiento, utilizar en el campo <IDENTIFICADOR> de la siguiente consulta:

```
mysql> SELECT event_name
AS Stage, TRUNCATE(TIMER_WAIT/1000000000000,6) AS Duration
FROM performance_schema.events_stages_history_long
WHERE NESTING_EVENT_ID=<IDENTIFICADOR>;
```

```
mysql> SELECT event_name
    -> AS Stage, TRUNCATE(TIMER_WAIT/1000000000000,6) AS Duration
    -> FROM performance_schema.events_stages_history_long
    -> WHERE NESTING_EVENT_ID=26;
+----------------------------------------------+----------+
| Stage                                        | Duration |
+----------------------------------------------+----------+
| stage/sql/starting                           |   0.0000 |
| stage/sql/Executing hook on transaction begin. |   0.0000 |
| stage/sql/starting                           |   0.0000 |
| stage/sql/checking permissions               |   0.0000 |
| stage/sql/checking permissions               |   0.0000 |
| stage/sql/Opening tables                     |   0.0000 |
| stage/sql/init                               |   0.0000 |
| stage/sql/System lock                        |   0.0000 |
| stage/sql/optimizing                         |   0.0000 |
| stage/sql/statistics                         |   0.0000 |
| stage/sql/preparing                          |   0.0000 |
| stage/sql/executing                          |   2.4777 |
| stage/sql/end                                |   0.0000 |
| stage/sql/query end                          |   0.0000 |
| stage/sql/waiting for handler commit         |   0.0000 |
| stage/sql/closing tables                     |   0.0000 |
| stage/sql/freeing items                      |   0.0002 |
| stage/sql/cleaning up                        |   0.0000 |
+----------------------------------------------+----------+
18 rows in set (0.00 sec)
```

Para esta consulta, se puede observar cómo 2.47 segundos (prácticamente el 100% del tiempo total de la consulta) se han dedicado a su ejecución. Si no fuese así, querría decir que parte del tiempo se dedica, p.e., a estar esperando a otras tareas de MySQL o del sistema operativo. En ese caso, sería conveniente buscar en la documentación de MySQL información sobre cómo reducir el tiempo en esas partes.

Como nota final, el esquema de rendimiento debería desactivarse en sistemas de producción ya que ralentiza significativamente el rendimiento de MySQL, haciendo empeorar la eficiencia de forma notable. Tanto el esquema de rendimiento como el comando Explain están pensados para ser utilizados en entornos de prueba.

4 Ejercicio

IMPORTANTE: Antes de comenzar, eliminar los índices e histogramas que se hayan creado.

En este ejercicio se trabajará en grupos de 2 personas. Utilizando la base de datos *Employees*, cada miembro de la pareja tendrá que elegir uno de los siguientes enunciados:

A. Obtener el máximo salario que se cobra en el departamento "Production".

B. Obtener la fecha de nacimiento del empleado más joven en la categoría "Engineer".

Para cada enunciado, se debe realizar lo siguiente:

1) Implementar la consulta sin utilizar técnicas optimización y obtener métricas de rendimiento: coste de los diferentes pasos y desglose de tiempos.
2) Aplicar técnicas de optimización a la consulta. Obtener métricas de rendimiento de la consulta optimizada y compararlas con la versión original. La consulta optimizada debe ejecutarse, al menos, un 30% más rápido que su versión sin optimizar. Si no se llega a este nivel, explicar por qué y qué pruebas se han realizado.

Para cada enunciado, se debe entregar lo siguiente en un único fichero de texto:

- Consulta sin optimizar.
- Métricas de rendimiento de la consulta sin optimizar.
- Comandos utilizados para optimizar la consulta.
- Métricas de rendimiento de la consulta optimizada y comparación con las métricas de la versión sin optimizar (p.e. cuántos pasos menos/mas hay, cuánto más rápido es en %, …).