

## ADBMS

### Exp4

Aim: To implement Query Monitor

Kartik Jolapara  
60004200107  
B1

ADBMS  
Exp4

Page No. 1  
Date

Aim: Implement Query Monitor (Query Execution plan, query statistics)

Theory: with the DB Query Monitor get information availability and the performance of the database as well as the execution time of the database queries. It can help you identify probable causes of performance degradation of your business applications. If the execution time of the queries are higher than usual it can indicate problem with the database.

Tasks can perform:

- ① Compare the execution times of SQL queries.
- ② Identify poor performing queries.
- ③ Allow to delete rows of query results.
- ④ Prevent major outages.
- ⑤ Compare the outputs.

In short it keeps an eye on important queries. The monitor dashboard is structured to give you an overview of the important part of the monitor.

⊛ Optimizing queries with 'EXPLAIN'

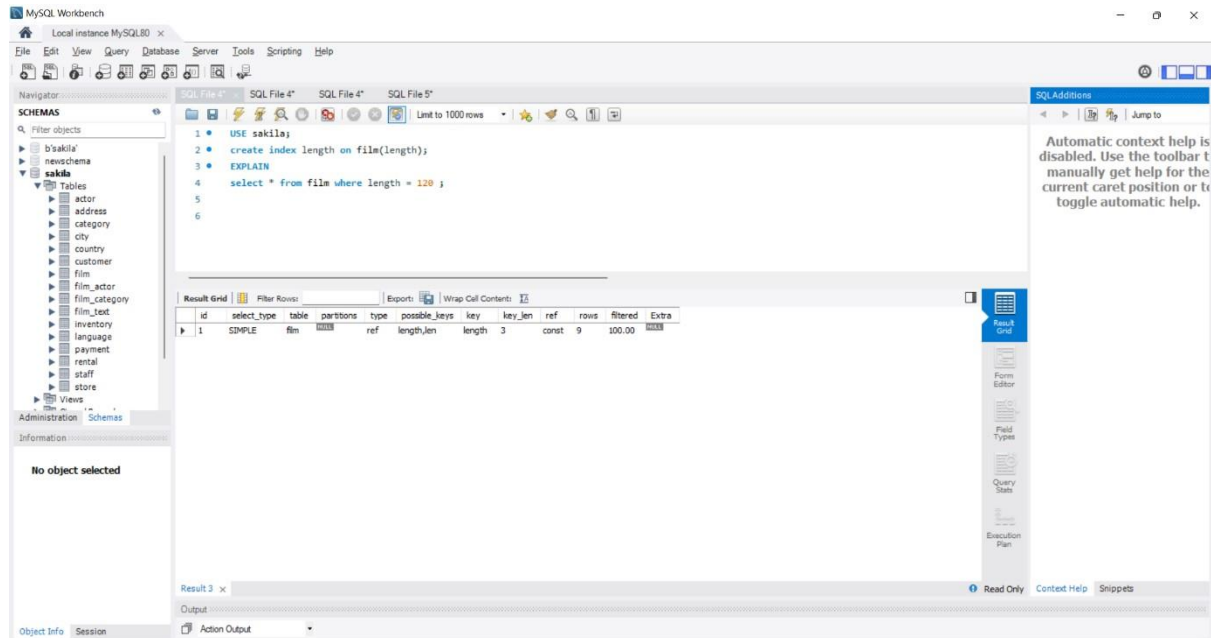
The 'EXPLAIN' statement provides information about how MySQL executes the statements.

- ① Explain works with SELECT, DELETE, UPDATE, INSERT & REPLACE
- ② Explain is useful for examining queries, involving partitioned tables
- ③ When EXPLAIN is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan.
- ④ For SELECT statement, explain produces additional execution plan information that can be displayed using SHOW WARNINGS
- ⑤ With Explain you can also see where you should add indexes to tables so that the statement executes faster by using indexes to find rows.
- ⑥ The optimizer trace may sometimes provide information complementary to that of EXPLAIN.
- ⑦ EXPLAIN can also be used to obtain information about the columns in a table

Conclusion: Hence, every Monitoring was implemented successfully in MySQL.

## Output:

### 1) SELECT QUERY



The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

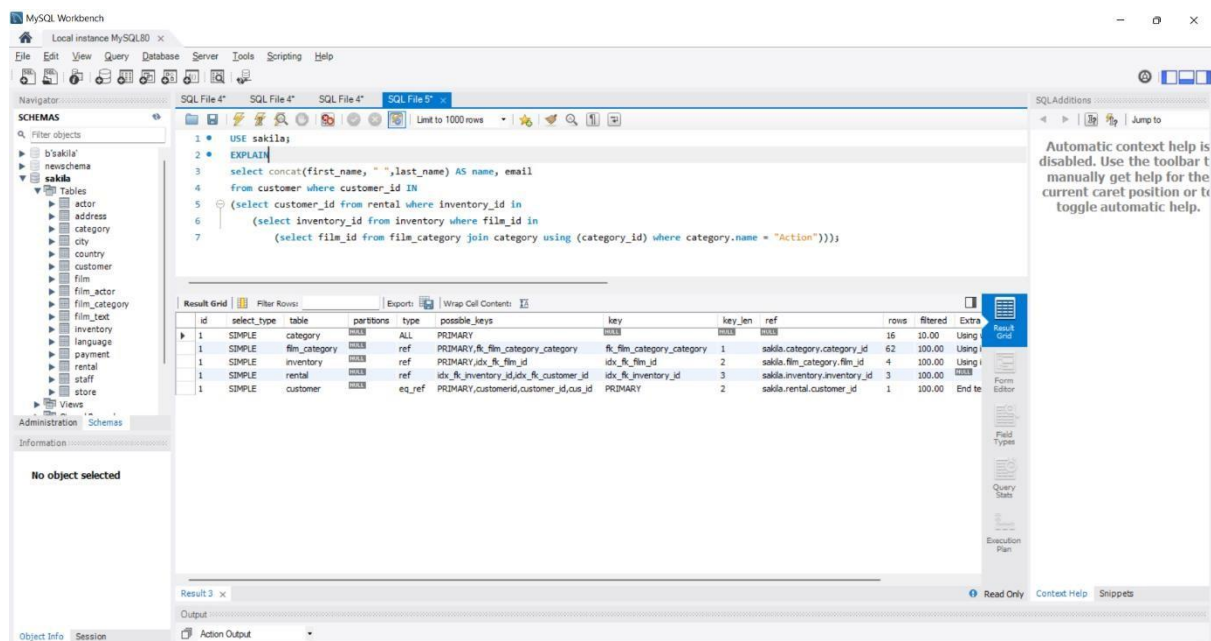
```
1 • USE sakila;
2 • create index length on film(length);
3 • EXPLAIN
4 select * from film where length = 120 ;
5
6
```

The Result Grid shows the execution plan for the query:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	film		ref	length	length	3	const	9	100.00	

The right sidebar shows the SQLAdditions panel with a message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

### 2) NESTED



The screenshot shows the MySQL Workbench interface. The SQL editor contains the following nested query:

```
1 • USE sakila;
2 • EXPLAIN
3 select concat(first_name, " ",last_name) AS name, email
4 from customer where customer_id IN
5 (select customer_id from rental where inventory_id in
6 (select inventory_id from inventory where film_id in
7 (select film_id from film_category join category using (category_id) where category.name = "Action"))));
```

The Result Grid shows the execution plan for the query:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	customer		eq_ref	PRIMARY, customerid	customer_id	4	sakila.customer_id	1	100.00	Using index
1	SIMPLE	inventory		ref	PRIMARY, idx_fk_inventory_id	idx_fk_inventory_id	4	sakila.inventory.inventory_id	3	100.00	Using index
1	SIMPLE	film_category		ref	PRIMARY, idx_fk_film_id	idx_fk_film_id	4	sakila.film_category.film_id	4	100.00	Using index
1	SIMPLE	category		ref	PRIMARY, idx_fk_category_id	idx_fk_category_id	4	sakila.category.category_id	62	100.00	Using index

The right sidebar shows the SQLAdditions panel with a message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

### 3) LEFT JOIN

The screenshot shows MySQL Workbench with a query window containing the following SQL code:

```
1 use sakila;
2
3 EXPLAIN
4 select stf.first_name, stf.last_name, ad.address, ad.district, ad.postal_code, ad.city_id
5 from staff stf
6 left join address ad
7 on stf.address_id = ad.address_id;
```

The result grid shows the execution plan for the query:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	stf		ALL					2	100.00	
1	SIMPLE	ad		eq_ref	PRIMARY	PRIMARY	2	sakila.stf.address_id	1	100.00	

The right sidebar contains a message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

### 4) RIGHT JOIN

The screenshot shows MySQL Workbench with a query window containing the following SQL code:

```
1 USE sakila ;
2
3 explain
4 select film.title , count(*) number_of_actors
5 from film film
6 right join film_actor film_act
7 on film.film_id = film_act.film_id
8 group by film.title
9 order by number_of_actors desc;
```

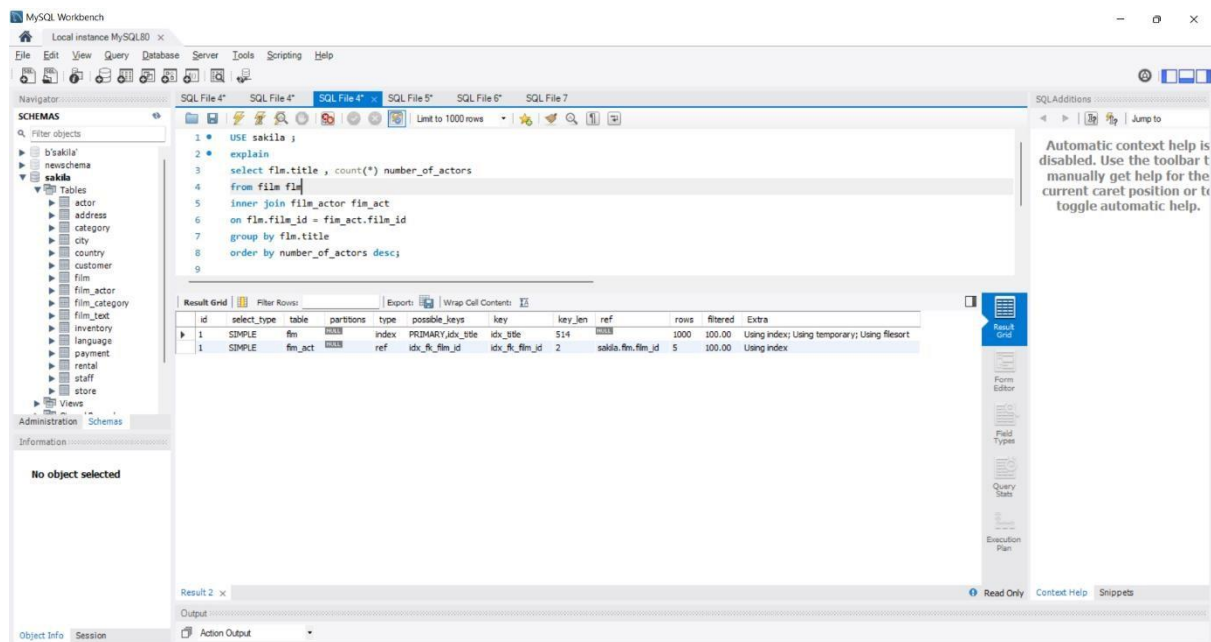
The result grid shows the execution plan for the query:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	film_act		index		idx_fk_film_id	2		5462	100.00	Using index; Using temporary; Using filesort
1	SIMPLE	film		eq_ref	PRIMARY, idx_title	PRIMARY	2	sakila.film_act.film_id	1	100.00	

The right sidebar contains a message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."



## 5) INNER JOIN

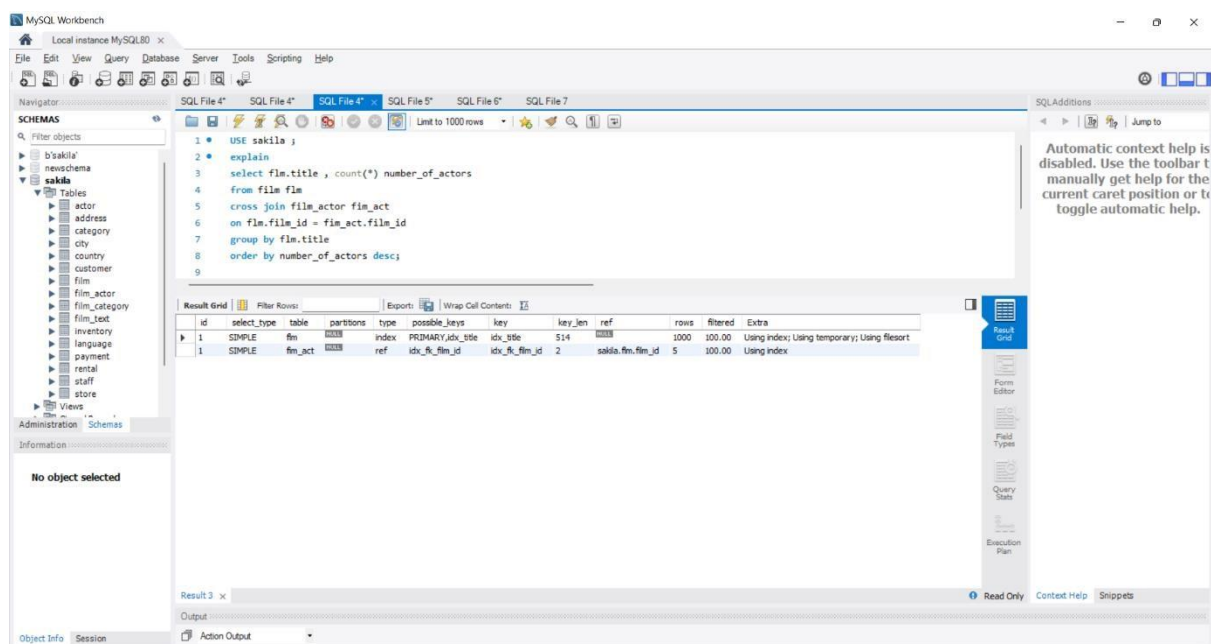


The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query is an INNER JOIN between the `film` and `film_actor` tables. The result grid shows the execution plan for the query.

```
1 • USE sakila ;
2 • explain
3 select film.title , count(*) number_of_actors
4 from film film
5 inner join film_actor film_act
6 on film.film_id = film_act.film_id
7 group by film.title
8 order by number_of_actors desc;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	film		index	PRIMARY,idx_title	idx_title	514		1000	100.00	Using index; Using temporary; Using filesort
1	SIMPLE	film_act		ref	idx_fk_film_id	idx_fk_film_id	2	sakila.film.film_id	5	100.00	Using index

## 6) CROSS JOIN



The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query is a CROSS JOIN between the `film` and `film_actor` tables. The result grid shows the execution plan for the query.

```
1 • USE sakila ;
2 • explain
3 select film.title , count(*) number_of_actors
4 from film film
5 cross join film_actor film_act
6 on film.film_id = film_act.film_id
7 group by film.title
8 order by number_of_actors desc;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	film		index	PRIMARY,idx_title	idx_title	514		1000	100.00	Using index; Using temporary; Using filesort
1	SIMPLE	film_act		ref	idx_fk_film_id	idx_fk_film_id	2	sakila.film.film_id	5	100.00	Using index

## Conclusion:

Thus, we implemented and studied query monitor