CS 432: Databases
Assignment 8: Query Optimization
The Eight

Q1)Optimize the query (having Like statement) for searching in the text-type column with the clause -
'UNION'. (Hint: Query can be stated as searching first_name's starting with 'A' OR last_name's
starting with 'B')

```
7   #A8 Q1
8   #Insert indexing to improve performance
9 • explain Select username from accounts where account_id in
10  (select account_id from user where city like "Sadar Bazar" union select account_id from vendor where city like "Sadar Bazar" );
```

Filter Rows: Export: Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	accounts	NULL	index	NULL	username_UNIQUE	402	NULL	80	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	user	NULL	eq_ref	PRIMARY	PRIMARY	4	func	1	11.11	Using where
3	DEPENDENT UNION	vendor	NULL	eq_ref	PRIMARY	PRIMARY	4	func	1	11.11	Using where
NULL	UNION RESULT	<union2,3>	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	Using temporary

```
7   #A8 Q1
8   #Insert indexing to improve performance
9 • Select username from accounts where account_id in
10  (select account_id from user where city like "Sadar Bazar" union select account_id from vendor where city like "Sadar Bazar" );
```

Filter Rows: Export: Wrap Cell Content:

Query_ID	Duration	Query
1	0.00009350	SHOW WARNINGS
2	0.00131525	Select username from accounts where account_id in (select account_id from user where city like "Sadar Bazar" union select account_id from vendor where city like "Sadar Bazar") LIMIT 0, 1000

Scanned rows=82

After optimization

```
7   #A8 Q1
8   #Insert indexing to improve performance
9 • explain Select username from accounts where account_id in
10  (select account_id from user where match(city) against("Sadar Bazar") union select account_id from vendor where match(city) against( "Sadar Bazar") );
```

Filter Rows: Export: Wrap Cell Content:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	accounts	NULL	index	NULL	username_UNIQUE	402	NULL	80	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	user	NULL	eq_ref	PRIMARY,account_id_idx1	PRIMARY	4	func	1	11.11	Using where
3	DEPENDENT UNION	vendor	NULL	eq_ref	PRIMARY,account_id_idx3	PRIMARY	4	func	1	11.11	Using where
NULL	UNION RESULT	<union2,3>	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	Using temporary

```
7    #A8 Q1
8    #Insert indexing to improve performance
9 •  Select username from accounts where account_id in
10   (select account_id from user where match(city) against("Sadar Bazar") union select account_id from vendor where match(city) against( "Sadar Bazar") );
```

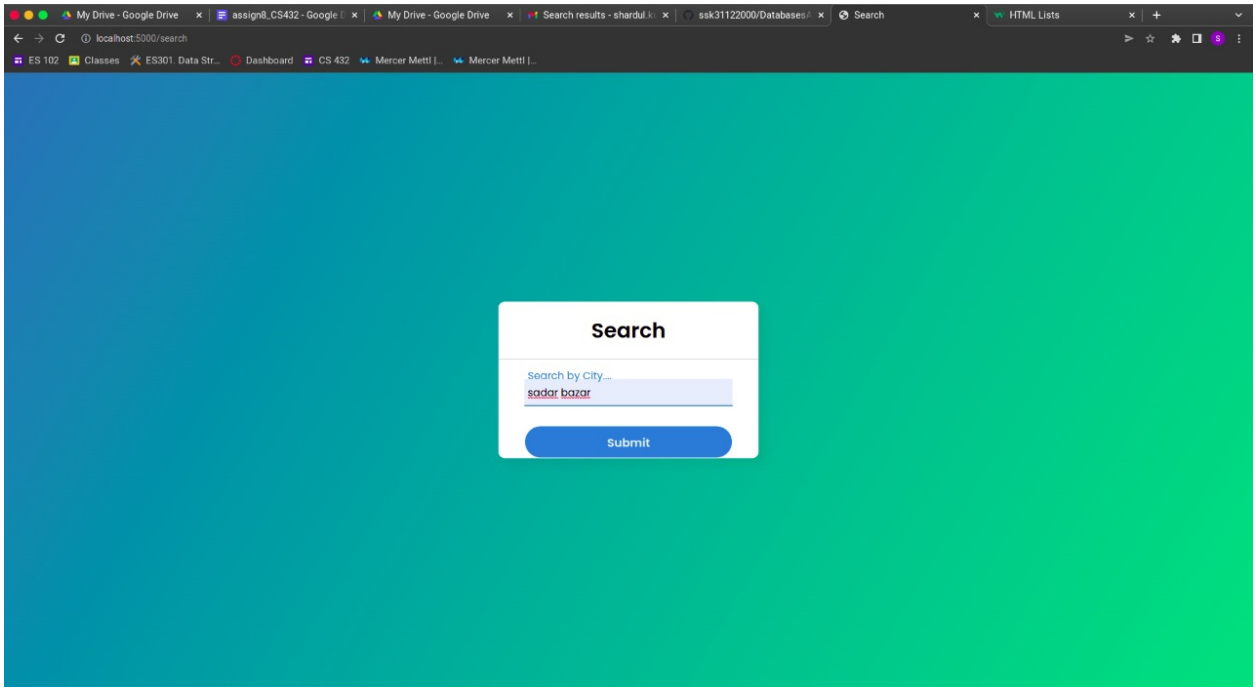
Filter Rows: Export: Wrap Cell Content:

Query_ID	Duration	Query
9	0.00083425	explain Select username from accounts where account_id in (select account_id from user where match(city) against("Sadar Bazar") union select account_id from vendor where match(city) against(…
10	0.00203875	Select username from accounts where account_id in (select account_id from user where match(city) against("Sadar Bazar") union select account_id from vendor where match(city) against(… LIMIT 0, 1000

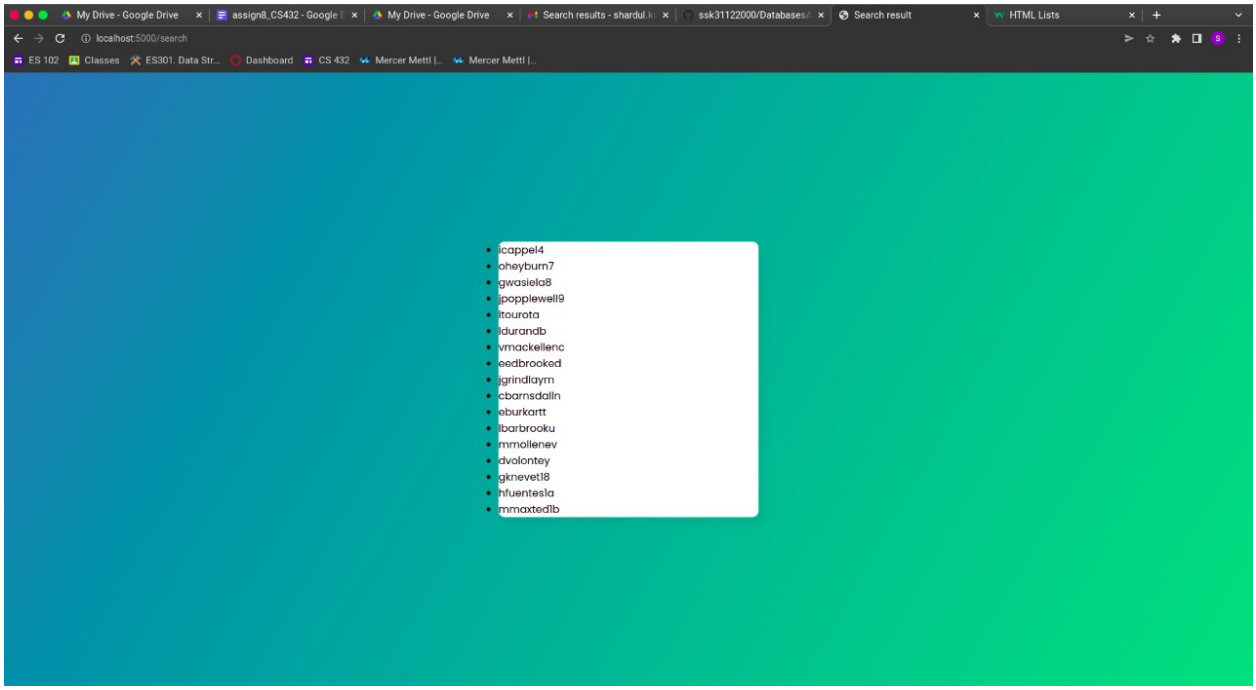
Scanned rows=82

We have applied full text indexing on the union of subqueries of searching city names

Search by the locality/city.



Users/Username in the city as result of before search.



(Q:2)Report the number of scans for optimizing the search through a column having text values with a chosen text pattern (Any text pattern of your choice). (5M)

- ❖ Finding the name of the city ending with “Bazar”

Original query:

select * from vendor WHERE city like '%Bazar%';

Optimised query:

ALTER TABLE vendor
ADD FULLTEXT(city);

select * from vendor WHERE MATCH(vendor.city) AGAINST ('Bazar')

Stats:

```
1 •  create database ecom;
2 •  use ecom ;
3 •  select * from vendor;
4 •  explain select * from vendor WHERE city like '%Bazar%';
5
```

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	vendor	NULL	ALL	NULL	NULL	NULL	NULL	20	11.11	Using where

account_id	warehouse_details	pincode	city	country	phone_number
53	0453 Carioca Parkway	281002	Sadar Bazar	India	5483257278
57	5 Knutson Way	281002	Sadar Bazar	India	1584164283
59	523 Porter Circle	281002	Sadar Bazar	India	8659122018
67	74 Corscot Way	281002	Sadar Bazar	India	8425774684

Number of scanned rows=20

Execution time= 0.0004510

Optimised Query:

```
1 •  create database ecom;
2 •  use ecom ;
3 •  ALTER TABLE vendor
4    ADD FULLTEXT(city);
5 •   explain select * from vendor WHERE MATCH(vendor.city) AGAINST ('Bazar')
```

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	vendor	NULL	fulltext	city	city	0	const	1	100.00	Using where; Ft_hints: sort

Execution time= 0.0001420

Number of scanned rows=1

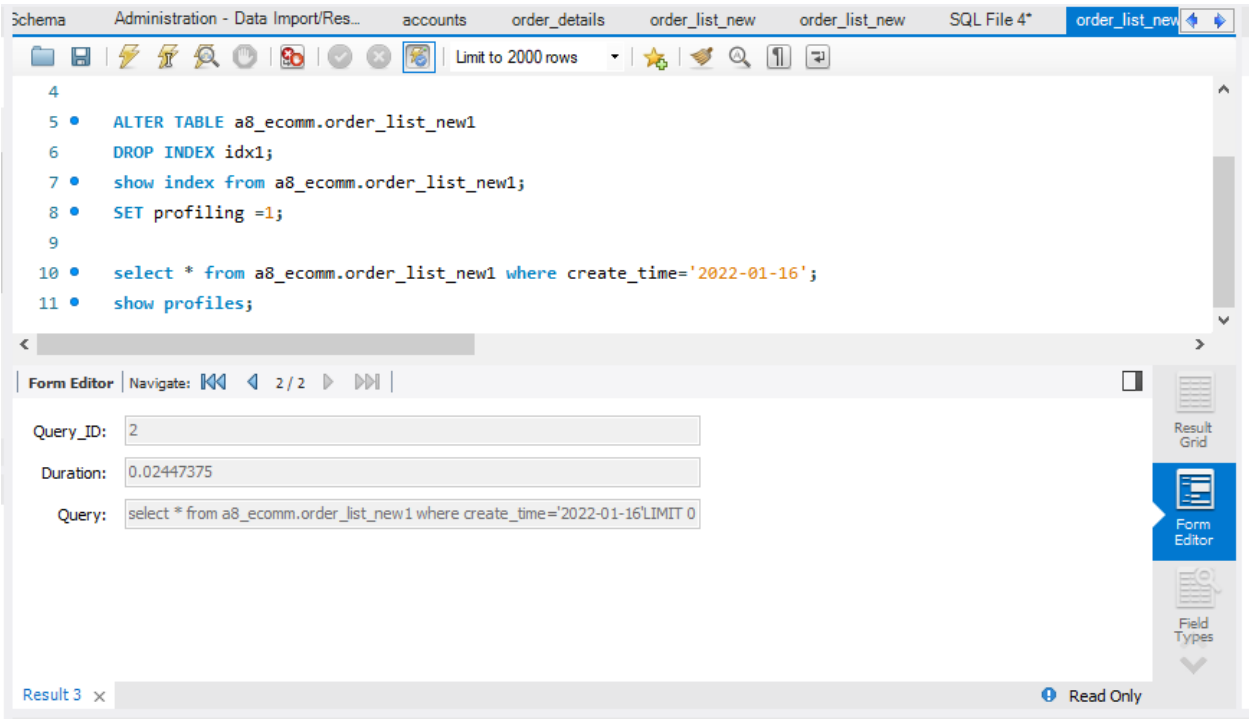
I have used full text search indexing to find the city column names ending with “bazar” to optimize the query .

Q3. Suppose there are limited data entries (Say 50 users' details), ideally it would be better if we
change the id’s data type to TINYINT (the signed range of TINYINT is from -128 to 127). Similarly, change the data type for either one from [phone_number, pin_code, address, or any
other specific to your statement].

```
12    #A8 Q3
13 •  alter table order_list modify order_quantity tinyint; #Since quantty will be less than 127 (assumed)
```

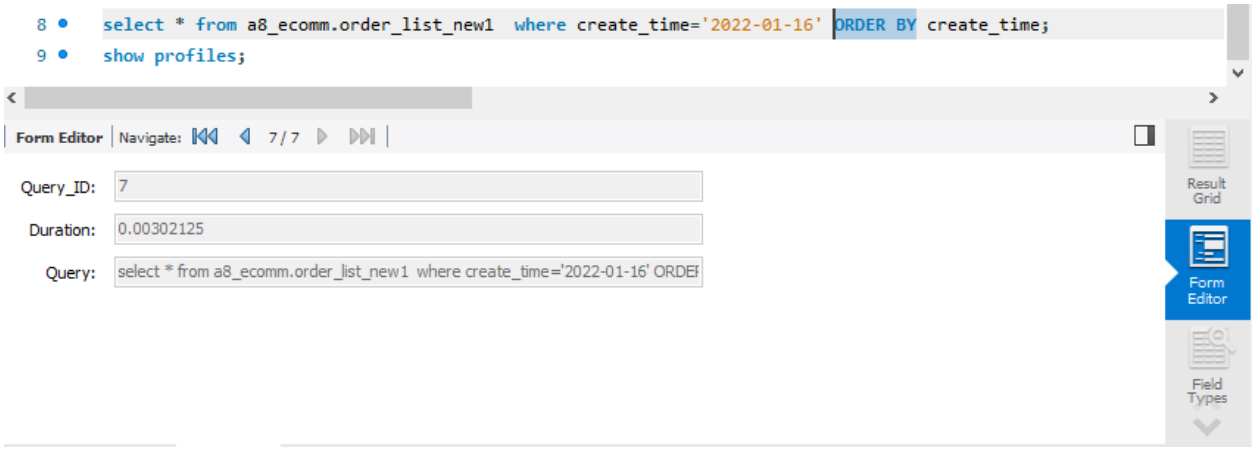
Q4.

Original:

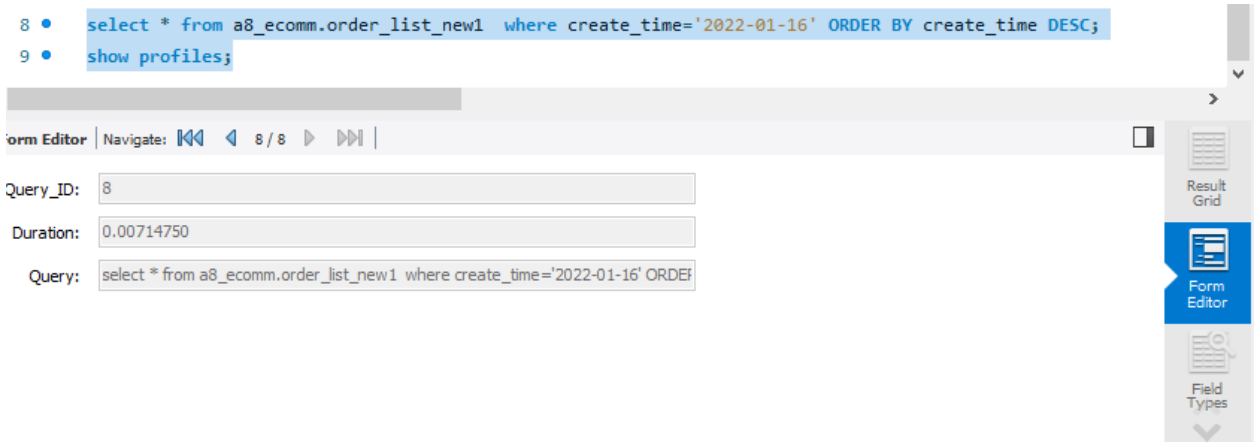


Some are below snapshots of trials with execution time for the various combinations with indexing type, groupby, order by etc operations.

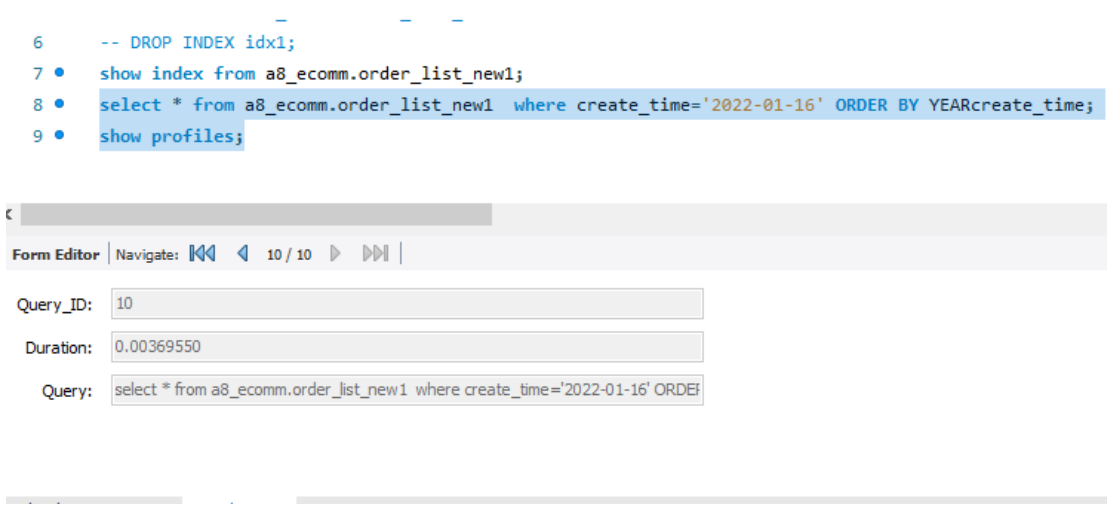
Order By Create_time Ascending



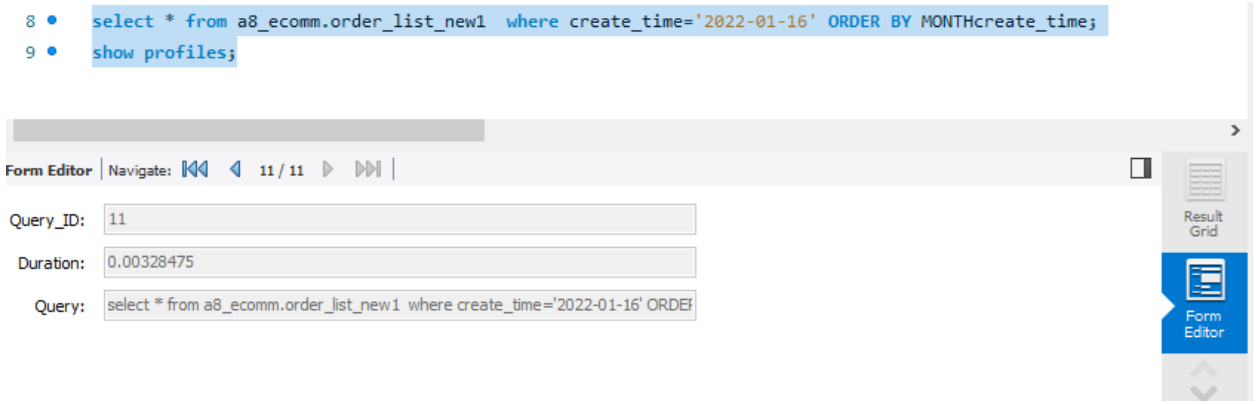
Order By Create_time Descending



ORDER BY YEARcreate_time;



ORDER BY MONTHcreate_time;



ORDER BY MONTHcreate_time, YEARcreate_time ;

```
8 •  select * from a8_ecomm.order_list_new1  where create_time='2022-01-16' ORDER BY MONTHcreate_time, YEARcreate_ti
9 •  show profiles;
```

Form Editor | Navigate: 14 / 14

Query_ID: 14

Duration: 0.00271600

Query: select * from a8_ecomm.order_list_new1 where create_time='2022-01-16' ORDEI

Result Grid | Form Editor

Unique index of order_id and orderby MONTHcreate_time, YEARcreate_time

```
3 •  create unique index order_idx on a8_ecomm.order_list_new1(order_id);
4
5 •  ALTER TABLE a8_ecomm.order_list_new1
6    DROP INDEX order_idx;
7 •  show index from a8_ecomm.order_list_new1;
8 •  select * from a8_ecomm.order_list_new1  where create_time='2022-01-16' ORDER BY MONTHcreate_time, YEARcreate_ti
9 •  show profiles;
```

Form Editor | Navigate: 15 / 15

Query_ID: 22

Duration: 0.00166050

Query: select * from a8_ecomm.order_list_new1 where create_time='2022-01-16' ORDEI

order list new1 21 | Result 22 ×

Read Only

Limit to 2000 rows

```
1 •  ALTER TABLE a8_ecomm.order_list_new1 engine memory;
2 •  create unique index order_idx on a8_ecomm.order_list_new1(order_id) using hash;
3 •  ALTER TABLE a8_ecomm.order_list_new1
4    DROP INDEX order_idx;
5 •  show index from a8_ecomm.order_list_new1;
6 •  select * from a8_ecomm.order_list_new1  where create_time='2022-01-16' ORDER BY MONTHcreate_time, YEARcreate_time;
7 •  show profiles;
```

Form Editor | Navigate: 15 / 15

Query_ID: 49

Duration: 0.00081200

Query: select * from a8_ecomm.order_list_new1 where create_time='2022-01-16' ORDEI

Form Editor

Concluding from above all the snapshots we do the following steps for optimizing query as follows:

1) First we alter the table by using engine memory by adding the following query:

```
ALTER TABLE a8_ecomm.order_list_new1 engine memory;
```

2) Then we create the unique index named “order_idx” using the column order_id by indexing method of btree.

```
create unique index order_idx on a8_ecomm.order_list_new1(order_id) using btree;
```

3) Our main query is finding certain date, for optimizing we split the column of create_time (date) into dd (day) | mm(month)| y(year) seperated by columns. Year column of create_time is named as “YEARcreate_time” and Month column of create_time is named as “MONTHcreate_time”

```
select * from a8_ecomm.order_list_new1  where create_time='2022-01-16' ORDER BY MONTHcreate_time, YEARcreate_time;
```

Further we order by YEARcreate_time, for search of create_time = 2022-01-16 and obtain the following results with minimum duration of execution.

We have iterated various options as follows for optimization.

Indexing Type + Indexing Method	GROUP BY	ORDER BY	DURATION
Btree + Unique	-	MONTHcreate_time	~0.00067125
Btree + Unique	-	YEARcreate_time	~0.00133925
Btree + Unique	-	MONTHcreate_time, YEARcreate_time	~0.00100400
Btree + Unique	-	create_time	~0.005
Hashing + Unique	-	MONTHcreate_time	~0.003
Hashing + Unique	-	YEARcreate_time	~0.006
Hashing + Unique	-	MONTHcreate_time, YEARcreate_time	~0.00081200

In the above iterations we cannot use fulltext since order_id is integer which is the primary key.

Q5.

Original Table	Added Intentional Null Values.

category_id	category	description
5001	Electronics	Endosc destru bile les
5002	Smartphones	Occlude leg vein NEC
5003	Fashion	Ins/rep 1 pul gen,rechrg
5004	Beauty	Limb shorten-metacar/car
5005	Books	Sm bowel endoscopy NEC
5006	Toys	Pericardiocentesis
5007	Home Decor	Bronchial operation NEC
5008	Kitchen	Dilat frontonasal duct
5009	Laptops	Removal brain stim lead
5010	Shoes	Suture anal laceration

category_id	category	description
5001	Electronics	Endosc destru bile les
5002	Smartphones	Occlude leg vein NEC
5003	Fashion	NULL
5004	Beauty	Limb shorten-metacar/car
5005	Books	Sm bowel endoscopy NEC
5006	Toys	Pericardiocentesis
5007	Home Decor	NULL
5008	Kitchen	NULL
5009	Laptops	Removal brain stim lead
5010	Shoes	Suture anal laceration

Query Before:

```
SELECT count(*) FROM a8_ecomm.cat_null;
```

Answer:

count(*)
10

Query Added:

```
SELECT count(*) FROM a8_ecomm.cat_null where description is NOT NULL;
```

Answer without including NULL:

count(*)
7

Q6.

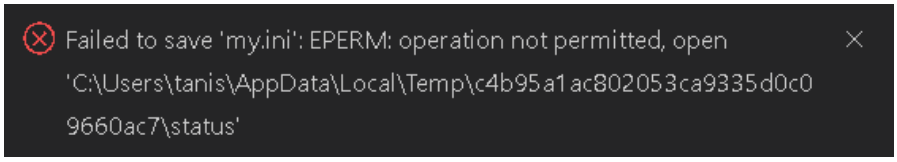
```
C:\Users\tanis>mysql -u root -p
```

```
mysql> SHOW STATUS LIKE "qcache%";
Empty set (0.14 sec)
```

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | NO    |
+------------------+-------+
1 row in set (0.11 sec)
```

As can be seen in the images above the query cache doesn't work. The query cache is deprecated as of MySQL 5.7.20, and is removed in MySQL 8.0.

Tried overwriting the MySQL config file, but failed.



The query cache has been disabled because it does not scale with high-throughput workloads on multi-core machines. For more information on this we can refer to a blog by Morgan tocker, click here for link.

Q7.

Queries used:

- select username, password from accounts left join admin on (accounts.account_id = admin.account_id) where admin.authorisation = 1;
- select username, password from accounts where account_id = (select account_id from admin where authorisation = 1 and admin.account_id = accounts.account_id);

320	16:37:54	select * from accounts left join admin on (accounts.account_id = admin.account_id) where admin.authorisation = 1 LIMIT 0, 1000	4 row(s) returned	0.015 sec / 0.000 sec
321	16:42:15	select username, password from accounts where account_id = (select account_id from admin where authorisation = 1 and admin.account_id = accounts.account_id) LIMIT 0, 1000	4 row(s) returned	0.109 sec / 0.000 sec

From the above operation, we see join operation consumes less time over subqueries.

Disadvantages of using multiple join operation:

1. Multiple Joins requires the server to do more work, thus consuming more time to retrieve it.
2. Different types of joins can create confusion, as each join gives out different results.
3. Usage of incorrect joins can result in performance degradation and inaccurate results.
4. Joints are not easy to read as subqueries.

Q.8

Unoptimized query:

```
7  explain SELECT city FROM
8  (SELECT user_id, total_price,city,create_time FROM user RIGHT JOIN order_details ON user.account_id=order_details.user_id
9  WHERE total_price=(SELECT MAX(total_price) FROM order_details))
10 AS temp
11 WHERE create_time=
12 (SELECT MAX(create_time) FROM user RIGHT JOIN order_details ON user.account_id=order_details.user_id WHERE total_price=(SELECT MAX(total_price) FROM order_details));
13
14 SET profiling=1;
15 SHOW profiles;
```

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

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	order_details	NULL	ALL	NULL	NULL	NULL	NULL	9	11.11	Using where
1	PRIMARY	user	NULL	eq_ref	PRIMARY,user_idx	PRIMARY	4	a8_ecom.order_details.user_id	1	100.00	NULL
4	SUBQUERY	order_details	NULL	ALL	NULL	NULL	NULL	NULL	9	11.11	Using where
4	SUBQUERY	user	NULL	eq_ref	PRIMARY,user_idx	PRIMARY	4	a8_ecom.order_details.user_id	1	100.00	Using index
5	SUBQUERY	order_details	NULL	ALL	NULL	NULL	NULL	NULL	9	100.00	NULL
3	SUBQUERY	order_details	NULL	ALL	NULL	NULL	NULL	NULL	9	100.00	NULL

Query_ID:	16
Duration:	0.00219850
Query:	SELECT city FROM (SELECT user_id, total_price,city,create_time FROM user RIGH

Number of scans=9+1+9+1+9+9=38
Execution time=0.00219850 sec

Optimized Query:

```
1  explain SELECT city FROM
2  (SELECT user_id, total_price,city,create_time FROM user RIGHT JOIN order_details ON user.account_id=order_details.user_id
3  WHERE total_price=(SELECT MAX(total_price) FROM order_details))
4  AS temp
5  ORDER BY create_time DESC LIMIT 1;
6
7  show profiles;
```

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

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	order_details	NULL	ALL	NULL	NULL	NULL	NULL	9	11.11	Using where; Using filesort
1	PRIMARY	user	NULL	eq_ref	PRIMARY,user_idx	PRIMARY	4	a8_ecom.order_details.user_id	1	100.00	NULL
3	SUBQUERY	order_details	NULL	ALL	NULL	NULL	NULL	NULL	9	100.00	NULL

Query_ID:	17
Duration:	0.00043400
Query:	SELECT city FROM (SELECT user_id, total_price,city,create_time FROM user RIGH

Number of scans=9+1+9=19
Execution time=0.00043400 sec