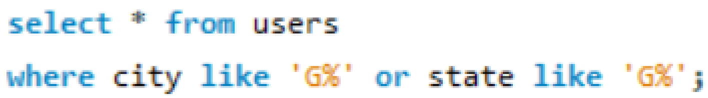
Assignment 8
CS432
Group Name: monkeDB

1. Database used: *iitgn_coviddata*
   Table: *users*
   Selected attributes: *state, city*
   Search: tuples in which the name of the state starts with G or the name of the city starts with G

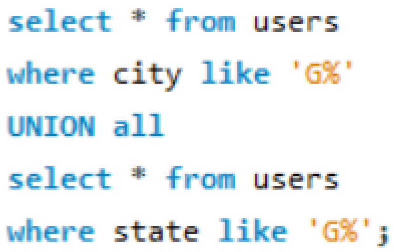
   When we use the search operation to get a set of tuples we use the OR operator for our desired conditions. In the following example, we have used OR operator to get the tuples that have the name of the city or state starting from 'G'. The OR operator in the WHERE clause makes the query inefficient as it runs a full table scan instead we can optimise this query by using the UNION operator.

   Unoptimized query:



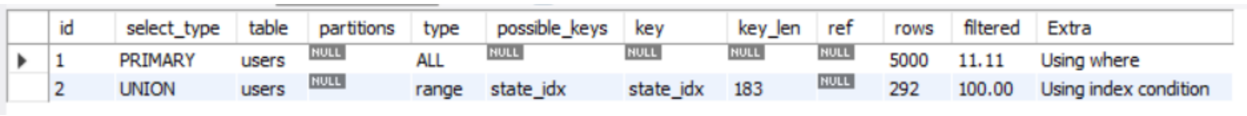
```
select * from users
where city like 'G%' or state like 'G%';
```

   Optimized Query:



```
select * from users
where city like 'G%'
UNION all
select * from users
where state like 'G%';
```

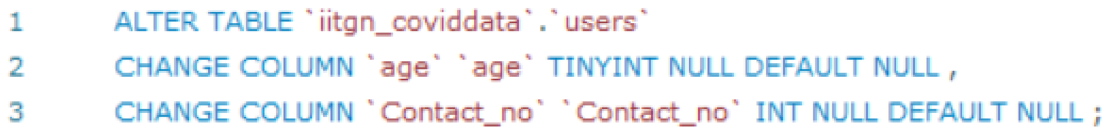
2. Following are the results for the TASK 1
   Number of scans for unoptimized query (use of OR): **5000**
   Number of scans for optimized query (use of UNION): **292**



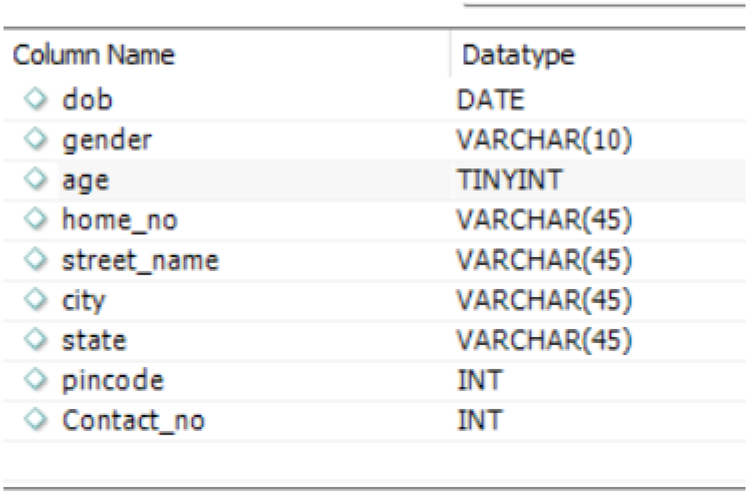
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	users	NULL	ALL	NULL	NULL	NULL	NULL	5000	11.11	Using where
	2	UNION	users	NULL	range	state_idx	state_idx	183	NULL	292	100.00	Using index condition

3. Initially in the database, the datatype of *age* was set as INT and the datatype of *pin code* and *contact number* was set to BIGINT. We changed the datatype of *age* to TINYINT and of *pin code* and *contact number* to INT. We are aware that age will be in the range of

-127 to 127 so we can perform this optimisation similarly we *pin code* and *contact number* can be stored in the range of INT which has a size smaller than BIGINT.



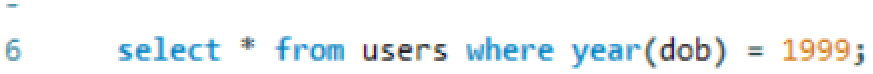
```
ALTER TABLE `iitgn_coviddata`.`users`
CHANGE COLUMN `age` `age` TINYINT NULL DEFAULT NULL ,
CHANGE COLUMN `Contact_no` `Contact_no` INT NULL DEFAULT NULL ;
```



Column Name	Datatype
◇ dob	DATE
◇ gender	VARCHAR(10)
◇ age	TINYINT
◇ home_no	VARCHAR(45)
◇ street_name	VARCHAR(45)
◇ city	VARCHAR(45)
◇ state	VARCHAR(45)
◇ pincode	INT
◇ Contact_no	INT

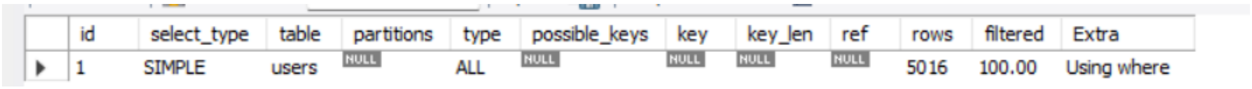
4. The *users* table had a date of birth (*dob*) column.

   Query:



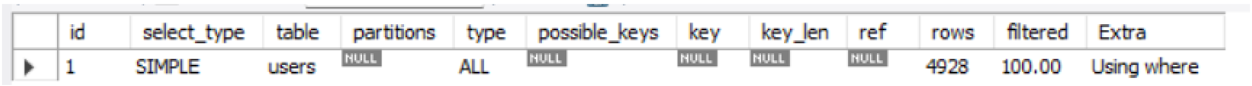
```
select * from users where year(dob) = 1999;
```

Datatype: TIMESTAMP



	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	users	NULL	ALL	NULL	NULL	NULL	NULL	5016	100.00	Using where

Datatype: DATE

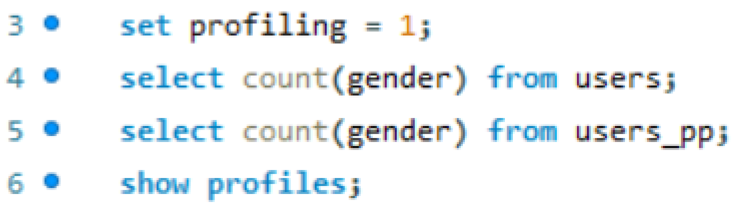


	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	users	NULL	ALL	NULL	NULL	NULL	NULL	4928	100.00	Using where

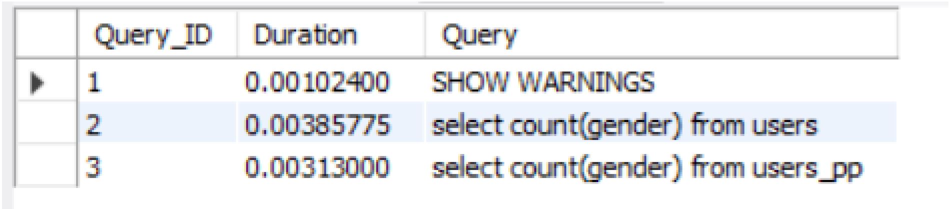
The number of scans has changed from 5016 to 4928. Changing the data type has impacted the search. TIMESTAMP data type has range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC, whereas DATE data type has range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59’. In the case of date of birth time component is not needed hence the better data type to use here is DATE.

5. We made a copy of the table *users* and named it *users_pp*. In this table, NULL values were added to the attribute *gender*.

   Null values: 1346
   Total: 5012



```
set profiling = 1;
select count(gender) from users;
select count(gender) from users_pp;
show profiles;
```



	Query_ID	Duration	Query
▶	1	0.00102400	SHOW WARNINGS
	2	0.00385775	select count(gender) from users
	3	0.00313000	select count(gender) from users_pp

   Output query_id 2: 5012
   Output query_id 3: 3666

   The above results obtained are as desired, query 1 returns **5012** tuples whereas query 2 returns **3666** tuples which means 1346 NULL values were detected. The duration of both the queries can also be seen in the above table. The duration of *users_pp* is lesser than the original query.

6. Caching queries are used to speed up the process, it also decreases the redundant execution of SQL queries and optimises the database. For our database, we can use the users, vaccine and covid_data tables for caching SELECT queries. The SQL query will look like the following:

```
SET query_cache_type=1
SET query_cache_size = 10M
SET query_cache_limit=256K
SET profiling = 1;
SELECT * from users
SELECT * from users

SELECT * from covid_data
SELECT * from covid_data

SELECT * from vaccine
SELECT * from vaccine

SHOW PROFILES;
```

Note that the queries need to be run twice to trigger query caching. The data is stored when the user runs the query the first time and when the query is executed for the second time it takes very less time to process. The performance increases drastically in the case of query caching.

7.

```
use iitgn_coviddata;
SELECT * from users as us
JOIN medical_history as mh where us.user_id = mh.user_id
and us.name LIKE "J%";
```

Joins merge records from two or more tables thus instead of going through multiple tables the query can be executed on a single table. Joins execute faster than other options like nested subqueries. Also joins have a better retrieval time of the query as compared to subquery.

In the case of multiple joins, the readability of the query decreases making it hard to understand. Multiple joins mean that the server has to do more work. Also, there are multiple types of joins making it difficult to choose the right join to get the desired output.

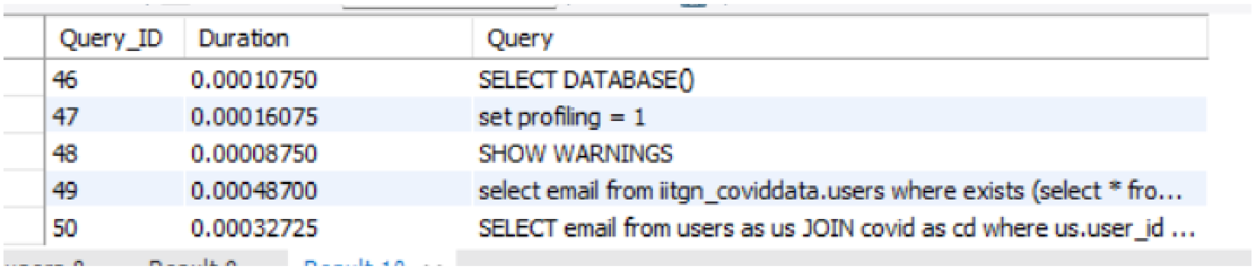
8. **Query 1**: Find the email of the patient with patient ID = 2

Nested Subquery:

```
select email from iitgn_coviddata.users
where exists
(select * from iitgn_coviddata.covid
where users.user_id = covid.user_id
and covid.patient_id = "2")
```

Optimized Query Using Join:

```
use iitgn_coviddata;
SELECT email from users as us
JOIN covid as cd where us.user_id = cd.user_id
and cd.patient_id = 2;
```



Query_ID	Duration	Query
46	0.00010750	SELECT DATABASE()
47	0.00016075	set profiling = 1
48	0.00008750	SHOW WARNINGS
49	0.00048700	select email from iitgn_coviddata.users where exists (select * fro...
50	0.00032725	SELECT email from users as us JOIN covid as cd where us.user_id ...

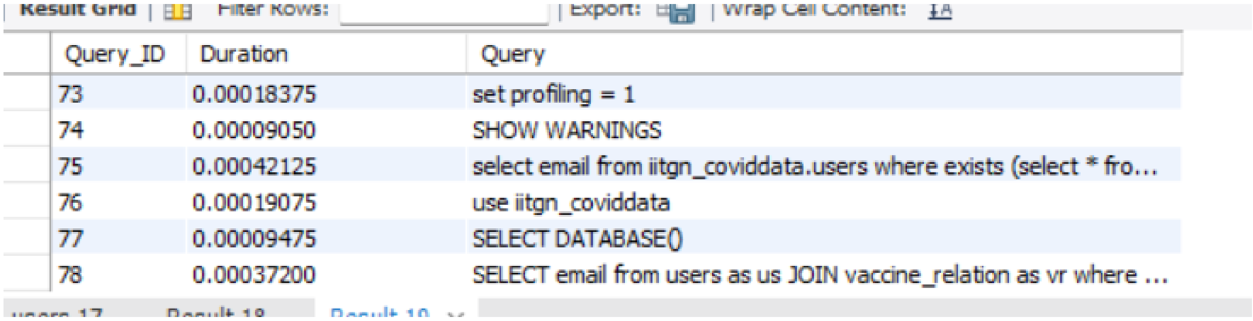
Query 2: Find the email of the person with vaccine ID = 2

Nested Subquery:

```
select email from iitgn_coviddata.users
where exists
(select * from iitgn_coviddata.vaccine_relation
where users.user_id = vaccine_relation.user_id
and vaccine_relation.vaccine_id = "2");
```

Optimized Query Using Join:

```
use iitgn_coviddata;
SELECT email from users as us
JOIN vaccine_relation as vr where us.user_id = vr.user_id
and vr.vaccine_id = 2;
```



Query_ID	Duration	Query
73	0.00018375	set profiling = 1
74	0.00009050	SHOW WARNINGS
75	0.00042125	select email from iitgn_coviddata.users where exists (select * fro...
76	0.00019075	use iitgn_coviddata
77	0.00009475	SELECT DATABASE()
78	0.00037200	SELECT email from users as us JOIN vaccine_relation as vr where ...

Query 3: Find the name of the student with QR ID = 3

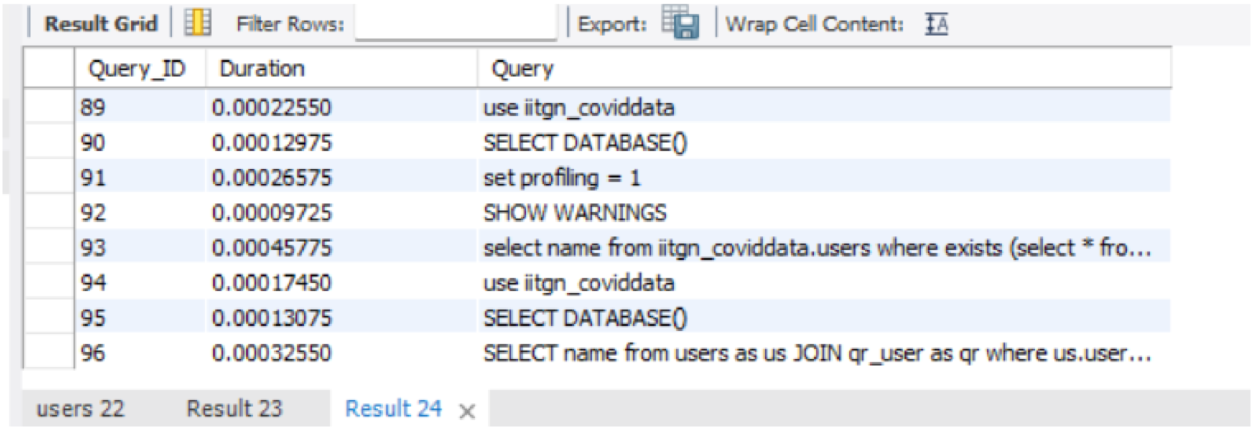
Nested Subquery:

```
select name from iitgn_coviddata.users
where exists
(select * from iitgn_coviddata.qr_user
where users.user_id = qr_user.Roll_no
and qr_user.qr_id = "3");
```

Optimized query using Join:

```
use iitgn_coviddata;
```

```
SELECT name from users as us
JOIN qr_user as qr where us.user_id = qr.roll_no
and qr.qr_id = 3;
```



Query_ID	Duration	Query
89	0.00022550	use iitgn_coviddata
90	0.00012975	SELECT DATABASE()
91	0.00026575	set profiling = 1
92	0.00009725	SHOW WARNINGS
93	0.00045775	select name from iitgn_coviddata.users where exists (select * fro...
94	0.00017450	use iitgn_coviddata
95	0.00013075	SELECT DATABASE()
96	0.00032550	SELECT name from users as us JOIN qr_user as qr where us.user...

In these three queries, we can see that the inner join works more efficiently and has a lower execution time.

Group Members and Contributions:

- Anas Ali: 20%
- Amish Raj: 15%
- Atul Patidar: 20%
- Jayesh Khanna: 15%
- Dhruvin Patel: 2.5%
- Jaydeep Ramnani: 2.5%
- Tanmay Sharma: 20%