E/15/385
Weerasinghe S.P.A.P.E

Department of Computer Engineering
University of Peradeniya
CO527 Advanced Database Systems

1) Use explain to analyse the outputs of following two simple queries which use only one table access.

(i)(ii)

```
ERROR 1054 (42S22): Unknown column 'Finance' in 'where clause'
mysql> explain SELECT * FROM departments WHERE dept_name = "Finance";
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table       | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | departments | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    9 |    11.11 | Using where |
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain SELECT * FROM departments WHERE dept_no = ’d002’;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '?d002?' at li
ne 1
mysql> explain SELECT * FROM departments WHERE dept_no = "d002";
+----+-------------+-------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table       | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | departments | NULL       | const | PRIMARY       | PRIMARY | 14      | const |    1 |   100.00 | NULL  |
+----+-------------+-------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

What conclusions you can draw from the results?

We can see that both queries are “Simple” which means it contains no subqueries or unions.

When considering access types (ii) is better than (i) because access type “All” is the worst than “const” [Access types sorted from the worst to the best: ALL, index, range, ref, eq_ref, const, system].

For (ii) dept_no can be taken as Primary key.

“rows” column shows that mysql have looked for 9 rows to execute (i) and , 1 row to execute (i). because dept_no is a possible primary key.

“filtered” show that (ii) has 100.00 unfileterd using where clause ,that is possible becaue the dept_no is “possible primary key” , so mysqk can use it in the execution.

2. Explain is especially important for join analysis because joins have much potential to increase the amount of server processing.

I. create table emplist select emp_no, first_name from employees;

```
mysql> create table emplist select emp_no,first_name from employees;
Query OK, 300024 rows affected (0.44 sec)
Records: 300024  Duplicates: 0  Warnings: 0
```

II. ***create table titleperiod select emp_no, title, datediff(to_date, from_date) as period FROM titles***

```
mysql> create table titleperiod select emp_no,title,datediff(to_date,from_date) as period from titles;
Query OK, 443308 rows affected (0.81 sec)
Records: 443308  Duplicates: 0  Warnings: 0

mysql>
```

Query:

```
select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and titleperiod.period>4000;
```

```
ERROR 1054 (42S22): Unknown column 'title_period.period' in 'field list'
mysql> select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and titleperiod.period>4000;
```

Explain

```
ERROR 1049 (42000): Unknown database 'database'
mysql> use company;
Database changed
mysql> explain select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and titleperiod.period>40
00;
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
| id | select_type | table       | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                              |
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
|  1 | SIMPLE      | titleperiod | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 443308 |    33.33 | Using where                                        |
|  1 | SIMPLE      | emplist     | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 300024 |    10.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
2 rows in set, 1 warning (0.03 sec)
```

Here from the explain out put “id”The query above contains no subqueries nor unions, so therefore the id for both rows is 1, as there is actually only 1 query.

For “select type” it's a simple query as it contains no subqueries or unions.

“table” gives tables involved as titleperiod and emplist.

“type” shows access type “ALL” which is good in accessing.

As you can see in the EXPLAIN, the table *titleperiod* is the first table accessed, using the ALL access_type, which means MySQL will scan the entire table, using no indexes, so it will go through all records. The *emplist* table is then accessed using the ALL access type which will also go through all the records.

possible_keys - The optional indexes MySQL can choose from, to look up for rows in the table. Some of the indexes in this list can be actually irrelevant, as a result of the execution order MySQL chose. In general, MySQL can use indexes to join tables. Said that, it won't use any index on in my query, as it will go through all of its rows anyway (except rows filtered by the WHERE clause).

key – Here no indexes used.This column indicates the actual index MySQL decided to use. It doesn't necessarily mean it will use the entire index, as it can choose to use only part of the index, from the left-most side of it.

filtered - The amount of rows unfiltered by the conditions in the WHERE clause. These rows will be joined to the table in the next row of the EXPLAIN plan. As mentioned previously, this is a guesstimate as well, so MySQL can be wrong with this estimation.

extra - Contains more information about the query processing. Let's look into the extras for our query:

using where - The WHERE clause is used to restrict which rows are fetched from the current table (*titleperiod)* and matched with the next table (*posts*).

When using a Block Nested-Loop Join, MySQL will, instead of automatically joining titleperiod, insert as many rows from emplist that it can into a join buffer and then scan the appropriate range of emplist once, matching each record in titleperiod to the join buffer

What could be the number of row combinations that MySQL would need to check?

From above output of the explain we can say that no indexes used so

,In general, MySQL can use indexes to join tables. Said that, it won't use any index on in my query, as it will go through all of its rows anyway (except rows filtered by the WHERE clause).

Therefore basically mysql has to go through all the rows, as shown in the explain output it is 443308 * 300024 respectively which outputs a very large number 133003039392.

3. Good columns to index are those that you typically use for searching, grouping,or sorting records. The query does not have any GROUP BY or ORDER BY clauses, but it does use columns for searching:

I. Create indexes on the columns used to join the tables. In the emplist table, emp_no can be used as a primary key because it uniquely identifies each row.

```
mysql> ALTER TABLE `emplist` ADD PRIMARY KEY `emp_no` (`emp_no`);
Query OK, 300024 rows affected (0.77 sec)
Records: 300024  Duplicates: 0  Warnings: 0

mysql> show index from emplist;
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| emplist |          0 | PRIMARY  |            1 | emp_no      | A         |      300024 |     NULL | NULL   |      | BTREE      |         |               |
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
1 row in set (0.00 sec)
```

II. In the titleperiod table, emp_no must be a non-unique index because multiple employees can share the same title:

```
mysql> ALTER TABLE `titleperiod` ADD INDEX `emp_no`(`emp_no`);
Query OK, 443308 rows affected (0.57 sec)
Records: 443308  Duplicates: 0  Warnings: 0

mysql> show index from titleperiod;
+-------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table       | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| titleperiod |          1 | emp_no   |            1 | emp_no      | A         |      443308 |     NULL | NULL   |      | BTREE      |         |               |
+-------------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
1 row in set (0.00 sec)
```

III. Analyse the outputs of EXPLAIN After creating the indexes

```
mysql> explain select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and titleperiod.period>40
00;
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type   | possible_keys | key     | key_len | ref                       | rows   | filtered | Extra       |
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
|  1 | SIMPLE      | titleperiod | NULL       | ALL    | emp_no        | NULL    | NULL    | NULL                      | 443308 |    33.33 | Using where |
|  1 | SIMPLE      | emplist     | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | company.titleperiod.emp_no |      1 |   100.00 | NULL        |
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

After adding the indexes the number of rows that mysql thinks which it needed is reduced to 1 in emplist table. We have a primary key in that table. And its access type have changed to eq_ref.As you can see in the EXPLAIN, the table *titleperiod* is the first table accessed, using the ALL access_type, which means MySQL will scan the entire table, using no indexes, so it will go through over 443308 records.
The *posts* table is then accessed using the eq_ref access type. Other than the

system and const types, eq_ref is the best possible join type. The database will access one row from this table for each combination of rows from the previous tables.

Is it possible to optimize the query execution further?If so, what can be done?

Yes futher optimizations can be done.

MySQL chooses to start with the *titleperiod* table.The EXPLAIN output shows that it has to go through 443308 rows, which is about all the rows in the table. That's nasty.

The Extra column indicates that MySQL tries to reduce the amount of rows it inspects using the WHERE clause, but it estimates to reduce those to 33.33%, which is still bad. A possible conclusion from this information is that the condition *titleperiod.emp_no* selective enough, so therefore millions of rows will be joined to the next table

Therefore for further optimization

ALTER TABLE `titleperiod` ADD INDEX `period` (`period`);

Adding the “period” in titleperiod table as non-unique index, which is the remaining attribute in the queries’ where clause

Which was not a index before.

```
mysql> ALTER TABLE `titleperiod` ADD INDEX `period` (`period`);
Query OK, 443308 rows affected (1.13 sec)
Records: 443308  Duplicates: 0  Warnings: 0

mysql> explain select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and titleperiod.period>40
00;
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type | possible_keys | key    | key_len | ref                   | rows   | filtered | Extra       |
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
|  1 | SIMPLE      | emplist     | NULL       | ALL  | PRIMARY       | NULL   | NULL    | NULL                  | 300024 |   100.00 | NULL        |
|  1 | SIMPLE      | titleperiod | NULL       | ref  | emp_no,period | emp_no | 4       | company.emplist.emp_no |      1 |    68.49 | Using where |
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.01 sec)
```

As you can see after adding that index the number of rows that might access have reduced. And we have now possible keys for titleperiod table which we can use for further optimization too. Also order of accessing tables also have changed, which is also a good impact interms of performance.

3. Query Rewriting Techniques

Assignment:
Write a brief analysis report (maximum 4 pages) summarizing the results of explain used in the examples in the lab sheet and explaining how explain helps in optimizing query execution.

Analysis Report about the use of EXPLAIN in optimizing query execution

Basically ,when we issue a query, the MySQL Query Optimizer tries to devise an optimal plan for query execution. But the basic strategy for execution of the command might not be the best option. Therefore we have to try to figure out what is current execution plan and how can we improve it. There are so many tools we can use to We can see information about the plan by prefixing the query with EXPLAIN. EXPLAIN is one of the most powerful tools at your disposal for understanding and optimizing troublesome MySQL queries.

```
ERROR 1049 (42000): Unknown database 'database'
mysql> use company;
Database changed
mysql> explain select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and titleperiod.period>40
00;
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
| id | select_type | table       | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                              |
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
|  1 | SIMPLE      | titleperiod | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 443308 |    33.33 | Using where                                        |
|  1 | SIMPLE      | emplist     | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 300024 |    10.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
2 rows in set, 1 warning (0.03 sec)
```

Here is the result for the EXPLAIN upon the initial query.

What is describes in these columns? Lets take is one by one.

id – a sequential identifier for each SELECT within the query (for when you have nested subqueries).Here the id is ‘1’.

select_type – the type of SELECT query. Possible values are:

We have got SIMPLE – the query is a simple SELECT query without any subqueries or UNIONs.

Other types of possible types are PRIMARY, DERIVED, SUBQUERY , UNION.

table – the table referred to by the row title first period and then emplist

type – how MySQL joins the tables used. This is one of the most insightful fields in the output because it can indicate missing indexes or how the query is written should be reconsidered. Here we can see

“All” – the entire table is scanned to find matching rows for the join. This is the worst join type and usually indicates the lack of appropriate indexes on the table.

possible_keys – shows the keys that can be used by MySQL to find rows from the table, though they may or may not be used in practice. In fact, this column can often help in optimizing queries since if the column is NULL, it indicates no relevant indexes could be found. Which is not a good feature here.

key_len – indicates the length of the index the Query Optimizer chose to use. There is no keys used therefore it is NULL here.

ref – Shows the columns or constants that are compared to the index named in the key column. MySQL will either pick a constant value to be compared or a column itself based on the query execution plan

rows – lists the number of records that were examined to produce the output. This Is another important column worth focusing on optimizing queries.

Extra – contains additional information regarding the query execution plan. Values such as “Using temporary”, “Using filesort”, etc. in this column may indicate a troublesome query.

This query is not performing well at all.So we need to optimize it to perform better one of the best way to optimize the query is to add “indexes”. Because as we can see there are no any index used in the initial query run.Here type is shown as “ALL” (which is the worst), which means MySQL was unable to identify any keys that can be used in the join and hence the possible_keys and key columns are null.

Accessing all rows and searching rows will lead to go through 443308 * 300024 = 133003039392 it will only increase exponentially as the database grows.

So we need to add “indexes ” to optimize the query. Good columns to index are those that you typically use for searching, grouping,
or sorting records. The query does not have any *GROUP BY* or *ORDER BY* clauses. But we don’t have any commands in our query.
It uses columns in the query

- The query uses emplist.emp_no and titleperiod.emp_no to match records between tables.
- The query uses titleperiod.period to cut down records that do not satisfy the condition.

So we add primary index to emplist.emp_no because it can use to uniquely identify the raws. TO create the connection between the two tables the emp_no of titletable also added as a index but it is added as non-unique index since there are many titles available with a emp_no.

```
mysql> explain select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and titleperiod.period>40
00;
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type   | possible_keys | key     | key_len | ref                       | rows   | filtered | Extra       |
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
|  1 | SIMPLE      | titleperiod | NULL       | ALL    | emp_no        | NULL    | NULL    | NULL                      | 443308 |    33.33 | Using where |
|  1 | SIMPLE      | emplist     | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | company.titleperiod.emp_no |      1 |   100.00 | NULL        |
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

After adding those indexes, the order of tables used by the query has changed and the number of raws accessed by the query is443308 which is way better than without indeces scenario.

Can we further improve the performance?

We will add the “period” as non unique index in the titleperiod table.This column was used in the initial query but not indexed**.**

```
mysql> ALTER TABLE `titleperiod` ADD INDEX `period` (`period`);
Query OK, 443308 rows affected (1.13 sec)
Records: 443308  Duplicates: 0  Warnings: 0

mysql> explain select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and title
00;
+----+-------------+-------------+------------+------+---------------+--------+---------+---------------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type | possible_keys | key    | key_len | ref                       | rows   | filtered | Extra       |
+----+-------------+-------------+------------+------+---------------+--------+---------+---------------------------+--------+----------+-------------+
|  1 | SIMPLE      | emplist     | NULL       | ALL  | PRIMARY       | NULL   | NULL    | NULL                      | 300024 |   100.00 | NULL        |
|  1 | SIMPLE      | titleperiod | NULL       | ref  | emp_no,period | emp_no | 4       | company.emplist.emp_no    |      1 |    68.49 | Using where |
+----+-------------+-------------+------------+------+---------------+--------+---------+---------------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.01 sec)
```

After adding that index the number of raws to be executed has decreased to 300024 which has increased the performance and the access “type” of titleperiod has changed into “ref” which is way better than “ALL”.

Upto now we have optimized the query very well. Can we optimize it further?

Use of **straight join**. Used select straight_join to force the oder of the two tables to be changed.And used the explain to check the result.

```
mysql> explain select straight_join emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod where emplist.emp_no=titleperiod.emp_no and titlepe
riod.period>4000;
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type | possible_keys | key    | key_len | ref                   | rows   | filtered | Extra       |
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
|  1 | SIMPLE      | emplist     | NULL       | ALL  | PRIMARY       | NULL   | NULL    | NULL                  | 300024 |   100.00 | NULL        |
|  1 | SIMPLE      | titleperiod | NULL       | ref  | emp_no,period | emp_no | 4       | company.emplist.emp_no |      1 |    68.49 | Using where |
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.01 sec)

mysql> explain select straight_join emplist.first_name,titleperiod.title,titleperiod.period from titleperiod,emplist where emplist.emp_no=titleperiod.emp_no and titlepe
riod.period>4000;
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type   | possible_keys | key     | key_len | ref                       | rows   | filtered | Extra       |
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
|  1 | SIMPLE      | titleperiod | NULL       | ALL    | emp_no,period | NULL    | NULL    | NULL                      | 443308 |    68.49 | Using where |
|  1 | SIMPLE      | emplist     | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | company.titleperiod.emp_no |      1 |   100.00 | NULL        |
+----+-------------+-------------+------------+--------+---------------+---------+---------+---------------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

From the above we can see that forcing the emplist table to execute first is the best choice.

Use of **Force Index.** Forced to use emp_no which is common to both tables.

```
mysql> explain select emplist.first_name,titleperiod.title,titleperiod.period from emplist,titleperiod force index(emp_no) where emplist.emp_no=titleperiod.emp_no and t
itleperiod.period>4000;
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type | possible_keys | key    | key_len | ref                   | rows   | filtered | Extra       |
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
|  1 | SIMPLE      | emplist     | NULL       | ALL  | PRIMARY       | NULL   | NULL    | NULL                  | 300024 |   100.00 | NULL        |
|  1 | SIMPLE      | titleperiod | NULL       | ref  | emp_no        | emp_no | 4       | company.emplist.emp_no |      1 |    33.33 | Using where |
+----+-------------+-------------+------------+------+---------------+--------+---------+-----------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

Here the number of row to access is 300024 which is way better improvement than the initial query.

Like this we can can conclude that . EXPLAIN is one of the most powerful tools at your disposal for understanding and optimizing troublesome MySQL queries.