**ADT Lab 4 (10pts)**

**Q1 Create clustered index**

**Q2 Create non-clustered index**

**Highlight the questions you are done with to green colour before you submit**

1. Download the files ADT Lab 4.docx, create\_table.sql, insert\_table.sql files from canvas.
2. Create a new database with your IU username (or use the one that you might have created in previous labs)
3. Run the sql commands in create\_table.sql to create new tables.
4. Run the sql commands in insert\_table.sql to populate all the tables.

In this tutorial we will be learning about indexing and how it can be used to optimize queries.

A clustered index and a non-clustered index are two types of indexes that can be created in a database to improve query performance. There are many types of indexes but the most commonly used ones are these two.

A clustered index is a type of index that determines the physical order of data in a table. In other words, the data rows of a table are stored on disk in the same order as the clustered index. There can be only one clustered index per table, as the data can only be physically ordered in one way. The clustered index is typically created on the primary key of the table, as the primary key is used most often in queries.

A non-clustered index, on the other hand, is a separate structure that contains a copy of some or all of the columns in a table and a pointer to the location of the corresponding data row. Unlike a clustered index, a non-clustered index does not affect the physical order of the data in the table. Instead, it provides a fast way to look up data by the indexed columns, without having to scan the entire table. There can be multiple non-clustered indexes in a table.

In summary, the key difference between clustered and non-clustered indexes is that a clustered index determines the physical order of data in a table, while a non-clustered index provides a fast way to look up data without affecting the physical order of the data.

How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more these costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. This is much faster than reading every row sequentially.

If you need further clarity refer- [Link](https://www.sqlshack.com/what-is-the-difference-between-clustered-and-non-clustered-indexes-in-sql-server/) and other sources on web

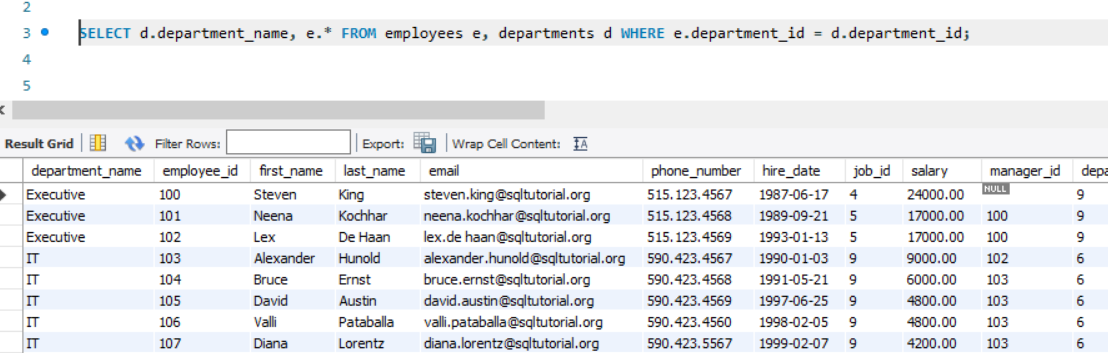
Questions

1. In our tutorial we will first analyse a query plan for a simple query.

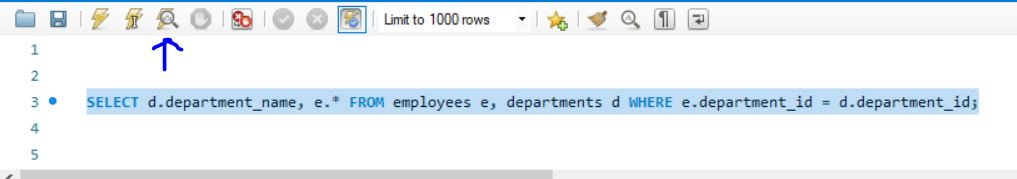
Once you run the create and insert commands run the below query.

SELECT d.department\_name, e.\* FROM employees e, departments d WHERE e.department\_id = d.department\_id;

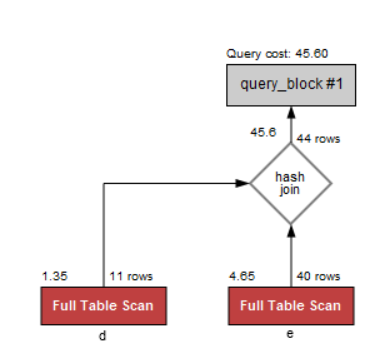
You should be able to see the output like this-



Now let us anayze the quey plan for this query. To do that select the query and click on the “Execute the EXPLAIN command” button



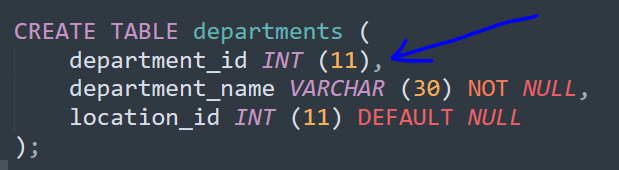
You should see something like this on your screen-



The “d” and “e” are the departments and employee tables respectively. This query plan says that both the tables departments and employees are scanned fully in order to execute this query.

Now is there a way to optimize this query?

Observe the create table statement for the department table-



We have not added the primary key in the create statement. This means we have not created any clustered index for department table.

MySQL does not support clustered indexes natively. However, you can achieve similar functionality to a clustered index by using a primary key. When you create a primary key on a table in MySQL, the database automatically creates an index on the primary key columns and uses it to enforce the unique constraint and improve query performance.

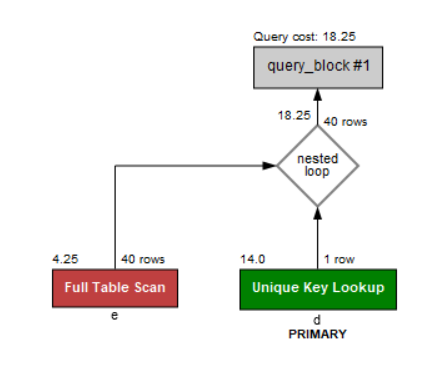
To add the index run the below query-

alter table departments add primary key (department\_id);

Now let’s run the same query again

SELECT d.department\_name, e.\* FROM employees e, departments d WHERE e.department\_id = d.department\_id;

The results are same but now when you analyse the query plan you see the following-



You can see the difference here- Only one row is being scanned from the department table for each row in employee table. Earlier for each row in employee table, all the entries in department table was being scanned.

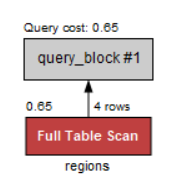
By adding index we have reduced the number of scans on the department table.

Now it’s your turn

Run the below query-

select \* from regions where region\_id=1;

Retrieve the query plan. It should be



Here to retrieve one row we are scanning 4 rows.

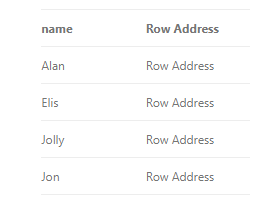
Create a primary key on region\_id similar to how we created for department\_id

Now retrieve the new query plan. It should scan a single row now after adding the index.

## Paste the screenshot of the updated query plan below

1. Now we will create a non-clustered index and see how it impacts the performance. Non-clustered index does not alter the way the data is stored in the disk. It simply keeps a mapping of the row value and its address-

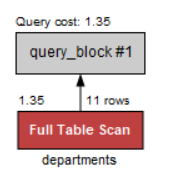
Example



Consider the departments table again. We can only create one clustered index per table. The basic rule of creating index is that- we create index on columns that are frequently used for joins or filters. So, there can be multiple such columns. But we have already created clustered index on department\_id. If need arises to create an index on some other column then non-clustered index is there for our rescue.

Analyse the query plan for this query-

select \* from departments where department\_name='IT';

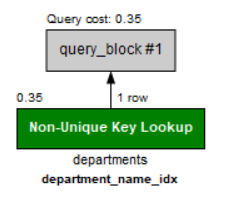


When we filter by the department\_name there are 11 rows scanned before giving us the results.

Now to create non-clustered index run the following-

create index department\_name\_idx on departments (department\_name);

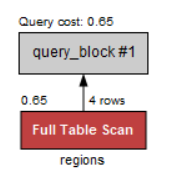
Now again analyse the query plan for the above query. You should get something like this-



Notice the reduction in query cost and now we scanned only 1 row to get the results.

Retrieve the query plan for below query-

select \* from regions where region\_name='Asia'



Similar to the above case where we created non clustered index on department\_name create a non-clustered index on region\_name and post a screenshot of the query plan

# Post your screenshot below