Summary – Day 10

Indexes:

- An index is a data structure that allows us to add indexes in the existing table. It enables you to improve the faster retrieval of records on a database table. It creates an entry for each value of the indexed columns. We use it to quickly find the record without searching each row in a database table whenever the table is accessed. We can create an index by using one or more columns of the table for efficient access to the records.
- When a table is created with a primary key or unique key, it automatically creates a special index named PRIMARY. We called this index as a clustered index. All indexes other than PRIMARY indexes are known as a non-clustered index or secondary index.

Need for Indexing in MySQL:

- Suppose we have a contact book that contains names and mobile numbers of the user. In this contact book, we want to find the mobile number of Martin Williamson. If the contact book is an unordered format means the name of the contact book is not sorted alphabetically, we need to go over all pages and read every name until we will not find the desired name that we are looking for. This type of searching name is known as sequential searching.

Syntax:

```
CREATE INDEX [index_name] ON [table_name] (column names)
```

Unique Index:

- Generally, we use the primary key constraint to enforce the uniqueness value of one or more columns. But, we can use only one primary key for each table. So if we want to make multiple sets of columns with unique values, the primary key constraint will not be used.
- MySQL allows another constraint called the UNIQUE INDEX to enforce the uniqueness of values in one or more columns. We can create more than one UNIQUE index in a single table, which is not possible with the primary key constraint.

Syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name (index_column1, index_column2,...);
```

Show Indexes:

We can get the index information of a table using the Show Indexes statement.

```
SHOW INDEXES FROM table_name;
```

Drop Index:

MySQL allows a DROP INDEX statement to remove the existing index from the table. To delete an index from a table, we can use the following query.

```
DROP INDEX index_name ON table_name
```

Views:

- A view is a database object that has no values. Its contents are based on the base table. It contains rows and columns similar to the real table. In MySQL, the View is a virtual table created by a query by joining one or more tables. It is operated similarly to the base table but does not contain any data of its own.
- The View and table have one main difference that the views are definitions built on top of other tables (or views). If any changes occur in the underlying table, the same changes reflected in the View also.
- Below query is used to create the view.

```
CREATE [OR REPLACE] VIEW view_name AS
SELECT columns
FROM tables
[WHERE conditions];
```

- We can change the logic in the view as well. To do that we use below query.

```
ALTER VIEW view_name AS
SELECT columns
FROM table
WHERE conditions;
```

- We can drop the existing VIEW by using the DROP VIEW statement.

```
DROP VIEW [IF EXISTS] view_name;
```

Purpose to use Views:

MySQL view provides the following advantages to the user.

- Simplify complex query:
  - It allows the user to simplify complex queries. If we are using the complex query, we can create a view based on it to use a simple SELECT statement instead of typing the complex query again.

- Increases the Re-usability:
  - We know that View simplifies the complex queries and converts them into a single line of code to use VIEWS. Such type of code makes it easier to integrate with our application. This will eliminate the chances of repeatedly writing the same formula in every query, making the code reusable and more readable.

- Help in Data Security:
  - It also allows us to show only authorized information to the users and hide essential data like personal and banking information. We can limit which information users can access by authoring only the necessary data to them.

- Enable Backward Compatibility:
    - A view can also enable the backward compatibility in legacy systems. Suppose we want to split a large table into many smaller ones without affecting the current applications that reference the table. In this case, we will create a view with the same name as the real table so that the current applications can reference the view as if it were a table.