Lab 12: View & Indexes

Objective(s):

1. To learn Views
2. To learn Indexes

1: 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.

Why we use View?

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.

We can create a new view by using the CREATE VIEW and SELECT statement. SELECT statements are used to take data from the source table to make a VIEW.

Syntax:

Following is the syntax to create a view in MySQL:

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

Parameters:

The view syntax contains the following parameters:

OR REPLACE: It is optional. It is used when a VIEW already exists. If you do not specify this clause and the VIEW already exists, the CREATE VIEW statement will return an error.

view_name: It specifies the name of the VIEW that you want to create in MySQL.

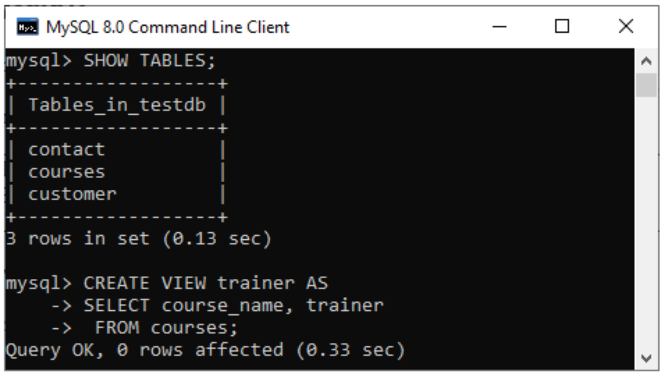
WHERE conditions: It is also optional. It specifies the conditions that must be met for the records to be included in the VIEW.

Example

Let us understand it with the help of an example. Suppose our database has a table **course**, and we are going to create a view based on this table. Thus, the below example will create a VIEW name "**trainer**" that creates a virtual table made by taking data from the table courses.

```
CREATE VIEW trainer AS
SELECT course_name, trainer
FROM courses;
```

Once the execution of the CREATE VIEW statement becomes successful, MySQL will create a view and stores it in the database.



```
mysql> SHOW TABLES;
+------------------+
| Tables_in_testdb |
+------------------+
| contact          |
| courses          |
| customer         |
+------------------+
3 rows in set (0.13 sec)

mysql> CREATE VIEW trainer AS
    -> SELECT course_name, trainer
    ->  FROM courses;
Query OK, 0 rows affected (0.33 sec)
```

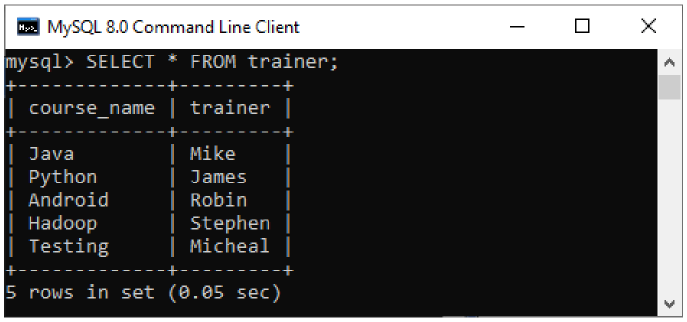
To see the created VIEW

We can see the created view by using the following syntax:

SELECT * **FROM** view_name;

Let's see how it looks the created VIEW:

SELECT * **FROM** trainer;



```
mysql> SELECT * FROM trainer;
+-------------+---------+
| course_name | trainer |
+-------------+---------+
| Java        | Mike    |
| Python      | James   |
| Android     | Robin   |
| Hadoop      | Stephen |
| Testing     | Micheal |
+-------------+---------+
5 rows in set (0.05 sec)
```

NOTE: It is essential to know that a view does not store the data physically. When we execute the SELECT statement for the view, MySQL uses the query specified in the view's definition and produces the output. Due to this feature, it is sometimes referred to as a virtual table.

MySQL Update VIEW

In MYSQL, the ALTER VIEW statement is used to modify or update the already created VIEW without dropping it.

Syntax:

Following is the syntax used to update the existing view in MySQL:

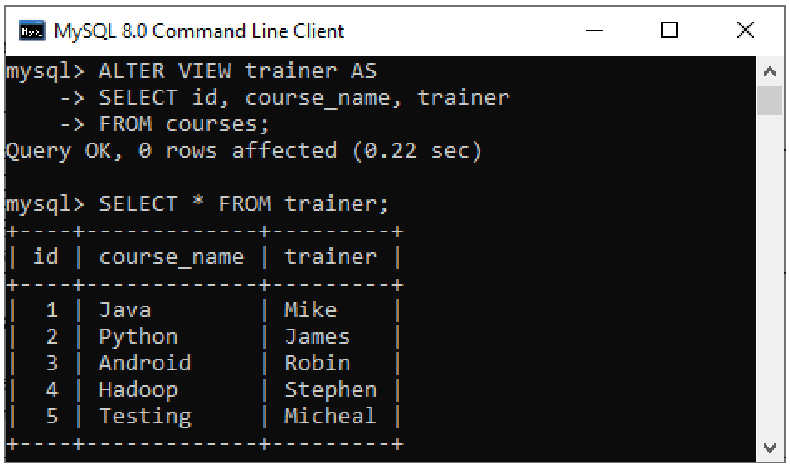
```
ALTER VIEW view_name AS
SELECT columns
FROM table
WHERE conditions;
```

Example:

The following example will alter the already created VIEW name "trainer" by adding a new column.

```
ALTER VIEW trainer AS
SELECT id, course_name, trainer
FROM courses;
```

Once the execution of the **ALTER VIEW** statement becomes successful, MySQL will update a view and stores it in the database. We can see the altered view using the SELECT statement, as shown in the output:



```
mysql> ALTER VIEW trainer AS
    -> SELECT id, course_name, trainer
    -> FROM courses;
Query OK, 0 rows affected (0.22 sec)

mysql> SELECT * FROM trainer;
+----+-------------+---------+
| id | course_name | trainer |
+----+-------------+---------+
|  1 | Java        | Mike    |
|  2 | Python      | James   |
|  3 | Android     | Robin   |
|  4 | Hadoop      | Stephen |
|  5 | Testing     | Micheal |
+----+-------------+---------+
```

MySQL Drop VIEW

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

Syntax:

The following is the syntax used to delete the view:

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

Parameters:

view_name: It specifies the name of the VIEW that we want to drop.

IF EXISTS: It is optional. If we do not specify this clause and the VIEW doesn't exist, the DROP VIEW statement will return an error.

Example:

Suppose we want to delete the view "**trainer**" that we have created above. Execute the below statement:

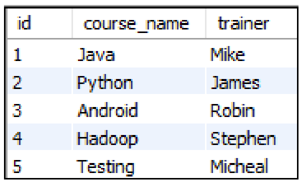
```
DROP VIEW trainer;
```

MySQL Create View with JOIN Clause

Here, we will see the complex example of view creation that involves multiple tables and uses a join clause.

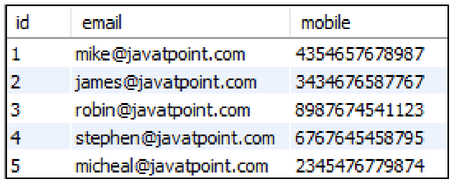
Suppose we have two sample table as shown below:

Table: course



id	course_name	trainer
1	Java	Mike
2	Python	James
3	Android	Robin
4	Hadoop	Stephen
5	Testing	Micheal

Table: contact

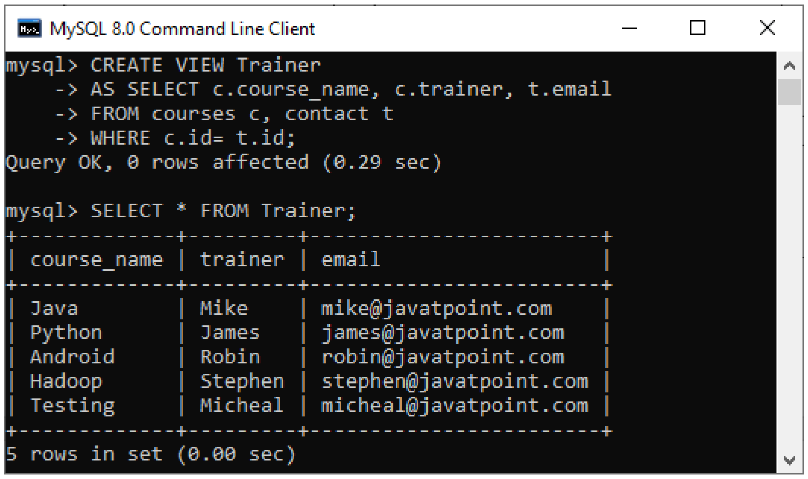


id	email	mobile
1	mike@javatpoint.com	4354657678987
2	james@javatpoint.com	3434676587767
3	robin@javatpoint.com	8987674541123
4	stephen@javatpoint.com	6767645458795
5	micheal@javatpoint.com	2345476779874

Now execute the below statement that will create a view Trainer along with the join statement:

```
CREATE VIEW Trainer
AS SELECT c.course_name, c.trainer, t.email
FROM courses c, contact t
WHERE c.id = t.id;
```

We can verify the view using the SELECT statement shown in the below image:



```
mysql> CREATE VIEW Trainer
    -> AS SELECT c.course_name, c.trainer, t.email
    -> FROM courses c, contact t
    -> WHERE c.id= t.id;
Query OK, 0 rows affected (0.29 sec)

mysql> SELECT * FROM Trainer;
+-------------+---------+-----------------------+
| course_name | trainer | email                 |
+-------------+---------+-----------------------+
| Java        | Mike    | mike@javatpoint.com   |
| Python      | James   | james@javatpoint.com  |
| Android     | Robin   | robin@javatpoint.com  |
| Hadoop      | Stephen | stephen@javatpoint.com |
| Testing     | Micheal | micheal@javatpoint.com |
+-------------+---------+-----------------------+
5 rows in set (0.00 sec)
```

To ensure the consistency of the view, you use the WITH CHECK OPTION clause when you create or modify the view.

2: 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.

To find the name and contact of the user from table **contactbooks**, generally, we used to execute the following query:

```
SELECT mobile_number FROM contactbooks WHERE first_name = 'Martin' AND last_name
 = 'John';
```

This query is very simple and easy. Although it finds the phone number and name of the user fast, the database searches entire rows of the table until it will not find the rows that you want. Assume, the contactbooks table contains **millions** of rows, then, without an index, the data retrieval takes a lot of time to find the result. In that case, the database indexing plays an important role in returning the desired result and improves the overall performance of the query.

MySQL CREATE INDEX Statement

Generally, we create an index at the time of table creation in the database. The following statement creates a table with an index that contains two columns col2 and col3.

```
CREATE TABLE t_index(
  col1 INT PRIMARY KEY,
  col2 INT NOT NULL,
  col3 INT NOT NULL,
  col4 VARCHAR(20),
  INDEX (col2,col3)
);
```

If we want to add index in table, we will use the CREATE INDEX statement as follows:

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

In this statement, **index_name** is the name of the index, **table_name** is the name of the table to which the index belongs, and the **column_names** is the list of columns.

Let us add the new index for the column col4, we use the following statement:

```
CREATE INDEX ind_1 ON t_index(col4);
```

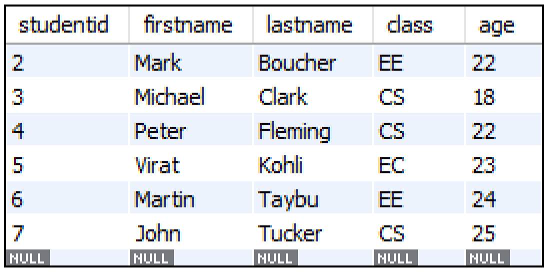
By default, MySQL allowed index type **BTREE** if we have not specified the type of index. The following table shows the different types of an index based on the storage engine of the table.

SN	Storage Engine	Index Type
1.	InnoDB	BTREE
2.	Memory/Heap	HASH, BTREE
3.	MYISAM	BTREE

Example

In this example, we are going to create a table **student** and perform the CREATE INDEX statement on that table.

Table Name: student

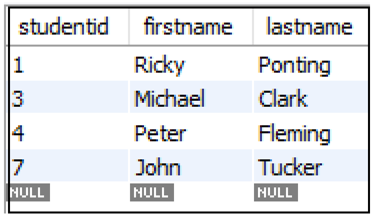


studentid	firstname	lastname	class	age
2	Mark	Boucher	EE	22
3	Michael	Clark	CS	18
4	Peter	Fleming	CS	22
5	Virat	Kohli	EC	23
6	Martin	Taybu	EE	24
7	John	Tucker	CS	25
NULL	NULL	NULL	NULL	NULL

Now, execute the following statement to return the result of the student whose **class** is **CS branch**:

```
SELECT studentid, firstname, lastname FROM student WHERE class = 'CS';
```

This statement will give the following output:



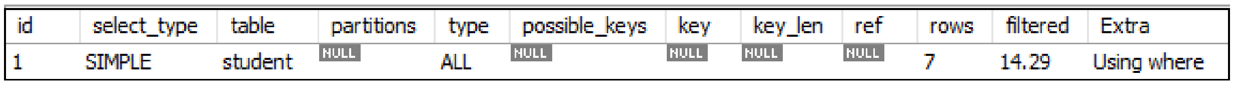
studentid	firstname	lastname
1	Ricky	Ponting
3	Michael	Clark
4	Peter	Fleming
7	John	Tucker
NULL	NULL	NULL

In the above table, we can see the four rows that are indicating the students whose class is the CS branch.

If you want to see how MySQL performs this query internally, execute the following statement:

```
EXPLAIN SELECT studentid, firstname, lastname FROM student WHERE class = 'CS';
```

You will get the output below. Here, MySQL scans the whole table that contains seven rows to find the student whose class is the CS branch.



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	student	NULL	ALL	NULL	NULL	NULL	NULL	7	14.29	Using where

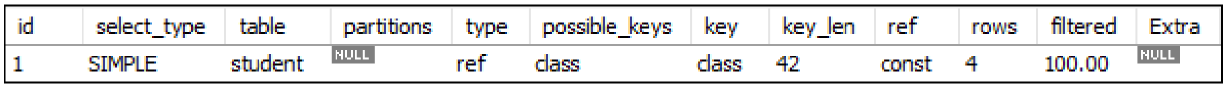
Now, let us create an index for a class column using the following statement.

```
CREATE INDEX class ON student (class);
```

After executing the above statement, the index is created successfully. Now, run the below statement to see how MySQL internally performs this query.

```
EXPLAIN SELECT studentid, firstname, lastname FROM student WHERE class = 'CS';
```

The above statement gives output, as shown below:



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	student	NULL	ref	class	class	42	const	4	100.00	NULL

In this output, MySQL finds four rows from the class index without scanning the whole table. Hence, it increases the speed of retrieval of records on a database table.

If you want to **show** the indexes of a table, execute the following statement:

```
SHOW INDEXES FROM student;
```

It will give the following output.

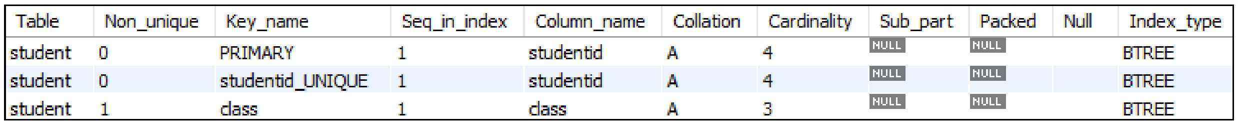


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
student	0	PRIMARY	1	studentid	A	4	NULL	NULL		BTREE
student	0	studentid_UNIQUE	1	studentid	A	4	NULL	NULL		BTREE
student	1	class	1	class	A	3	NULL	NULL		BTREE

MySQL 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 [algorithm_option | lock_option];
```

If we want to delete an index, it requires two things:

- First, we have to specify the name of the index that we want to remove.
- Second, name of the table from which your index belongs.

The Drop Index syntax contains two optional options, which are Algorithm and Lock for reading and writing the tables during the index modifications. Let us explain both in detail:

Algorithm Option

The algorithm_option enables us to specify the specific algorithm for removing the index in a table. The syntax of **algorithm_option** are as follows:

```
ALGORITHM [=] {DEFAULT | INPLACE | COPY}
```

The Drop Index syntax supports mainly two algorithms which are INPLACE and COPY.

COPY: This algorithm allows us to copy one table into another new table row by row and then DROP Index statement performed on this new table. On this table, we cannot perform an INSERT and UPDATE statement for data manipulation.

INPLACE: This algorithm allows us to rebuild a table instead of copy the original table. We can perform all data manipulation operations on this table. On this table, MySQL issues an exclusive metadata lock during the index removal.

Note: If you not defined the algorithm clause, MySQL uses the INPLACE algorithm. If INPLACE is not supported, it uses the COPY algorithm. The DEFAULT algorithm works the same as without using any algorithm clause with the Drop index statement.

Lock Option

This clause enables us to control the level of concurrent reads and writes during the index removal. The syntax of **lock_option** are as follows:

```
LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

In the syntax, we can see that the lock_option contains **four modes** that are DEFAULT, NONE, SHARED, and EXCLUSIVE. Now, we are going to discuss all the modes in detail:

SHARED: This mode supports only concurrent reads, not concurrent writes. When the concurrent reads are not supported, it gives an error.

DEFAULT: This mode can have the maximum level of concurrency for a specified algorithm. It will enable concurrent reads and writes if supported otherwise enforces exclusive mode.

NONE: You have concurrent read and write if this mode is supported. Otherwise, it gives an error.

EXCLUSIVE: This mode enforces exclusive access.

Example

First, execute the following command to show the indexes available in the table.

```
SHOW INDEXES FROM student;
```

It will give the following output.

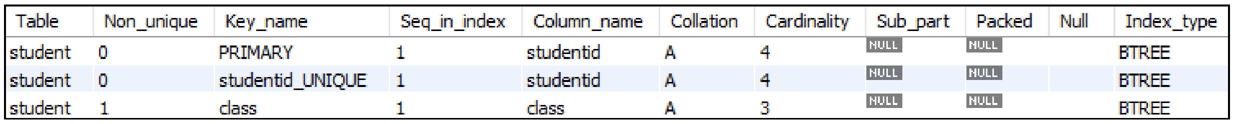


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
student	0	PRIMARY	1	studentid	A	4	NULL	NULL		BTREE
student	0	studentid_UNIQUE	1	studentid	A	4	NULL	NULL		BTREE
student	1	class	1	class	A	3	NULL	NULL		BTREE

In the output, we can see that there are three indexes available. Now, execute the following statement to removes the **class** index from table **student**.

```
DROP INDEX class ON student;
```

Again, execute the SHOW INDEXES statement to verify the index is removed or not. After performing this statement, we will get the following output, where only two indexes are available.

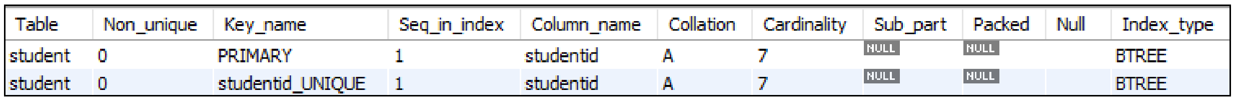


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
student	0	PRIMARY	1	studentid	A	7	NULL	NULL		BTREE
student	0	studentid_UNIQUE	1	studentid	A	7	NULL	NULL		BTREE

Example using Algorithm and Lock

The following statement drops the **age** index form the student table using an algorithm and a lock option.

```
DROP INDEX age ON student ALGORITHM = INPLACE LOCK = DEFAULT;
```

MySQL Drop PRIMARY Key Index

In some cases, the table contains a PRIMARY index that was created whenever you create a table with a primary key or unique key. In that case, we need to execute the following command because the PRIMARY is a reserved word.

```
DROP INDEX PRIMARY ON table_name;
```

To remove the primary key index from the student table, execute the following statement:

```
DROP INDEX PRIMARY ON student;
```

Lab Task(s):

Exercise

1. 1. Create a view to display records of deptno 10 from EMP table with the following attributes: empno, ename, sal, deptno.
2. Display all the records from view created in question 1.
3. Through view created in question 1, update the deptno of employee 200 from 10 to 20 and then display all the records from the same view and note which employee record is missing now, which was displayed in question 2.
4. Create any simple view with base table EMP in such a way that records can only be displayed but can never be manipulated through this view.(WITH CHECK OPTION)
5. Create a view which displays the ename, dname and sal of all the employees of deptno 20.
6. Create a view MY_VU based on the table EMP55 which does not exists in the schema. Now create the table EMP55 from EMP table having records of deptno=10. Now select all the records from the view MY_VU.
7. Create a copy of an employee table Emp2 with first_name col as an index.
8. Create an index of department_name col in department table.
9. Drop first_name index from Emp2 table.
10. Drop all indexes (Primary as well) from department_table. For that you need to write a query that first show all indexes of a table then write a query to drop indexes.

END