What is a Database?

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds.

Other kinds of data stores can also be used, such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those type of systems.

Nowadays, we use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as Foreign Keys.

A Relational DataBase Management System (RDBMS) is a software that –

- Enables you to implement a database with tables, columns and indexes.
- Guarantees the Referential Integrity between rows of various tables.
- Updates the indexes automatically.
- Interprets an SQL query and combines information from various tables.

RDBMS Terminology

Before we proceed to explain the MySQL database system, let us revise a few definitions related to the database.

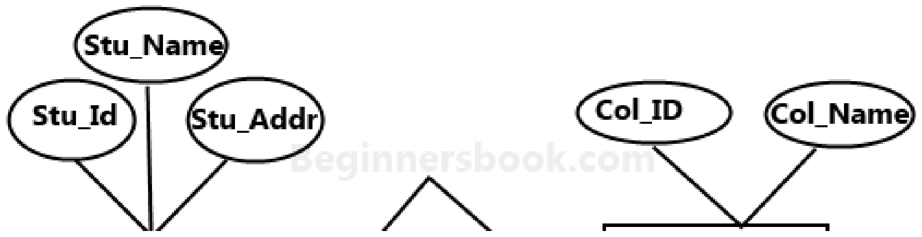
- **Database** – A database is a collection of tables, with related data.
- **Table** – A table is a matrix with data. A table in a database looks like a simple spreadsheet.
- **Column** – One column (data element) contains data of one and the same kind, for example the column postcode.
- **Row** – A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.
- **Redundancy** – Storing data twice, redundantly to make the system faster.
- **Primary Key** – A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row.
- **Foreign Key** – A foreign key is the linking pin between two tables.
- **Compound Key** – A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
- **Index** – An index in a database resembles an index at the back of a book.
- **Referential Integrity** – Referential Integrity makes sure that a foreign key value always points to an existing row.

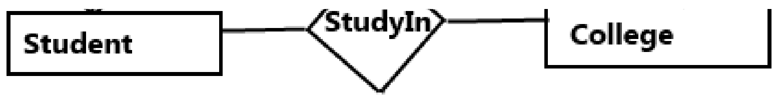
Database Design

In order build the database we have to design it first for that we have to plot the entire database system on a canvas using visualization tool.

There two approches of designing the MySQL database system and these are :-

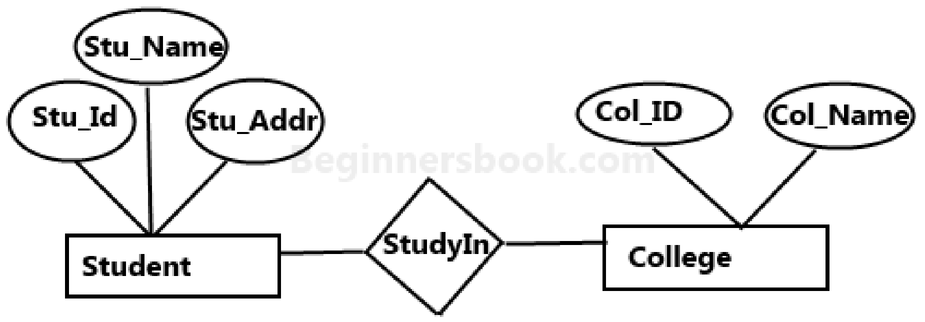
Entity Relationship (ER) diagram.





Sample E-R Diagram

Relational Schema



Sample E-R Diagram

Primary Key

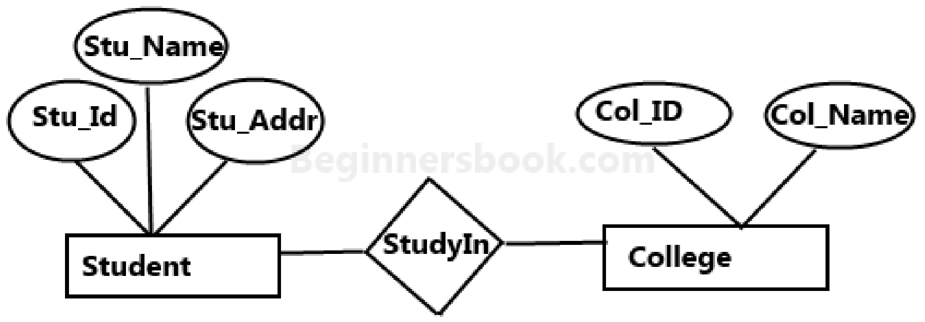
Column or set of the columns whose value exists and is unique for every record in the table called as **Primary Key**

Each table can have only one and only one primary key.

In the table we can not have 3 or 4 primary key.

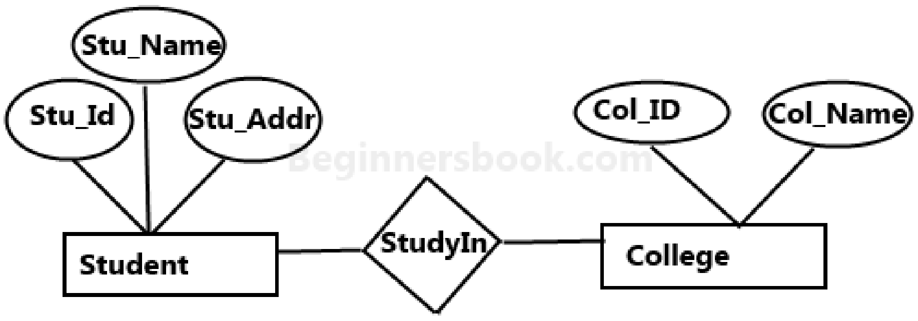
Example

In the below figure we can make the two column as Primary because there is duplicate data.



Sample E-R Diagram

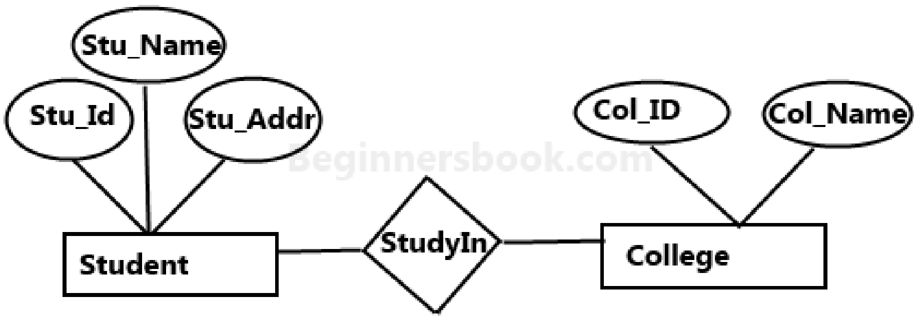
In the below way we can do the primary because they are containing unique values in each record and Not null as well.



Sample E-R Diagram

Primary key can not contains null values and values should be unique.

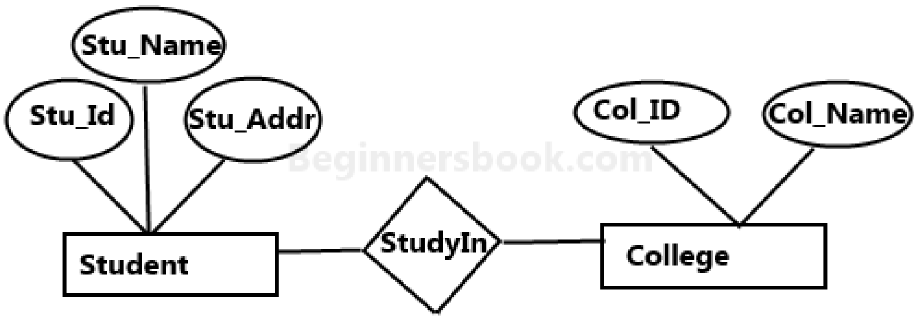
Not all the table that we work have the primary key.



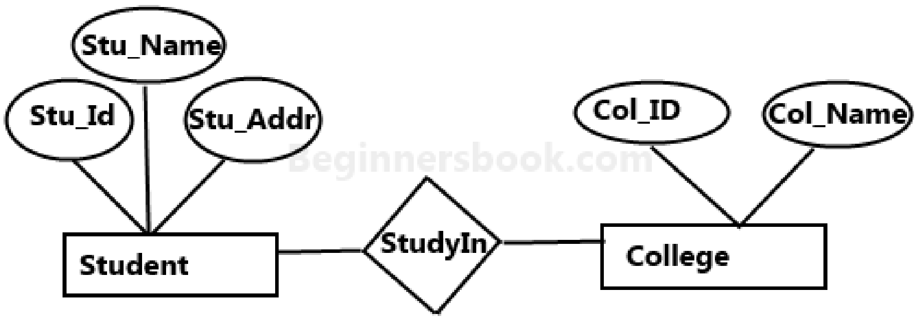
Sample E-R Diagram

Foreign Key

In order to link the two table on basis on their relationship and common data avalibilty so that we can maintain the relationship.

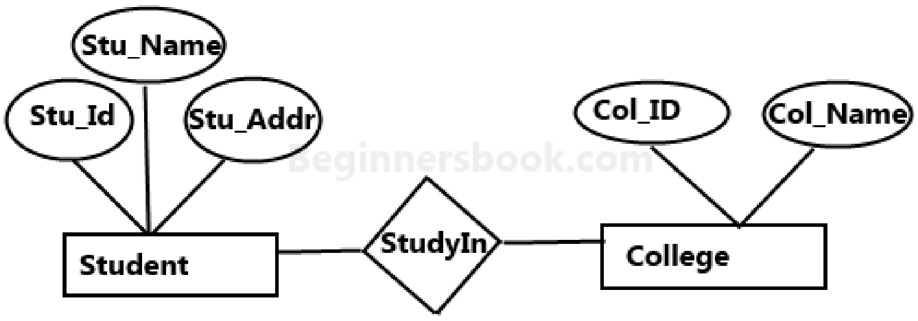


Sample E-R Diagram



Sample E-R Diagram

In the foreign key we can containing the missing values and duplicate value too.



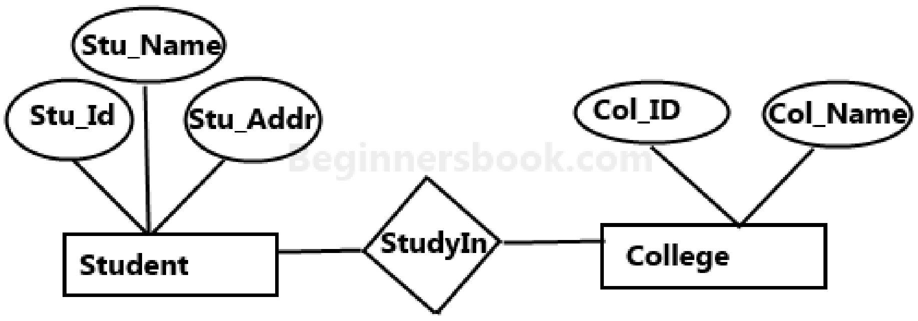
Sample E-R Diagram

Unique Key

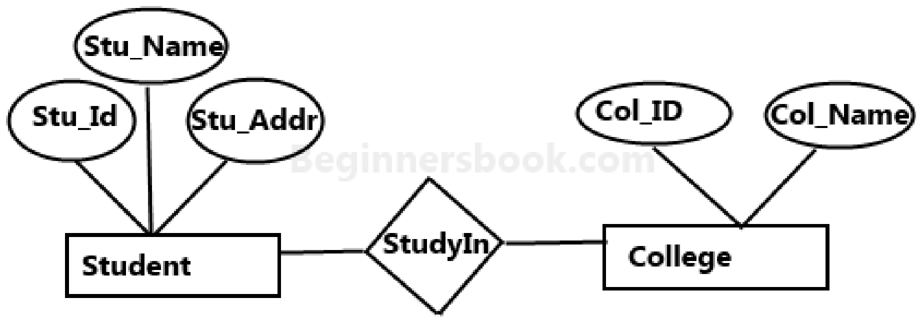
Let suppose we have table that contains the unique set of values inside records of MySQL database.Like Phone Number,password and Bank details etc.

In that we can define them as Primary key but all data is unique inside the table so we have special key for that is Unique Key.

Whenever we dont want to see the duplicate data inside the given field we have to use the unique key.



Sample E-R Diagram



Sample E-R Diagram

Data Types In SQL

We must have to specify the type of the data that will be inserted in each column of the table,different data type provide the diferent type of information that can be contained in specific column.

Numerical Data Type

String Data Type

In [ ]:

In [ ]:

In [ ]:

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

- **DDL** – Data Definition Language
- **DML** – Data Manipulation Language
- **DCL** – Data Control Language
- **TCL** - Transaction Control Language

DDL(Data Definition Language) :

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Examples of DDL commands:

- CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- DROP – is used to delete objects from the database.
- ALTER- is used to alter the structure of the database.
- TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT –is used to add comments to the data dictionary.
- RENAME – is used to rename an object existing in the database.

DML(Data Manipulation Language):

DML(Data Manipulation Language): The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. Examples of DML:

- INSERT – is used to insert data into a table.
- UPDATE – is used to update existing data within a table.
- DELETE – is used to delete records from a database table.

DCL(Data Control Language):

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions and other controls of the database system. Examples of DCL commands:

- GRANT-gives user's access privileges to the database.
- REVOKE-withdraw user's access privileges given by using the GRANT command.

TCL(transaction Control Language):

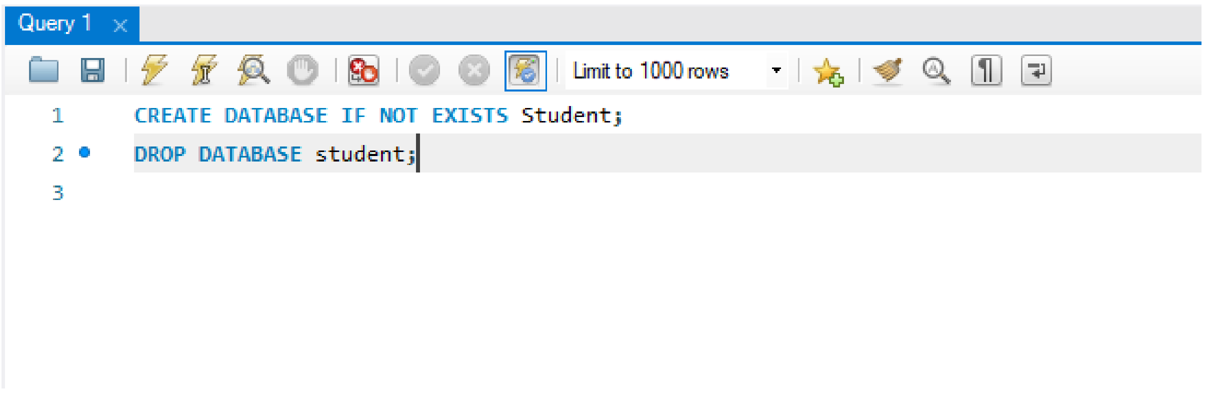
TCL commands deal with the transaction within the database. Examples of TCL commands:

- COMMIT– commits a Transaction.
- ROLLBACK– rollbacks a transaction in case of any error occurs.
- SAVEPOINT–sets a savepoint within a transaction.
- SET TRANSACTION–specify characteristics for the transaction.

In [ ]:

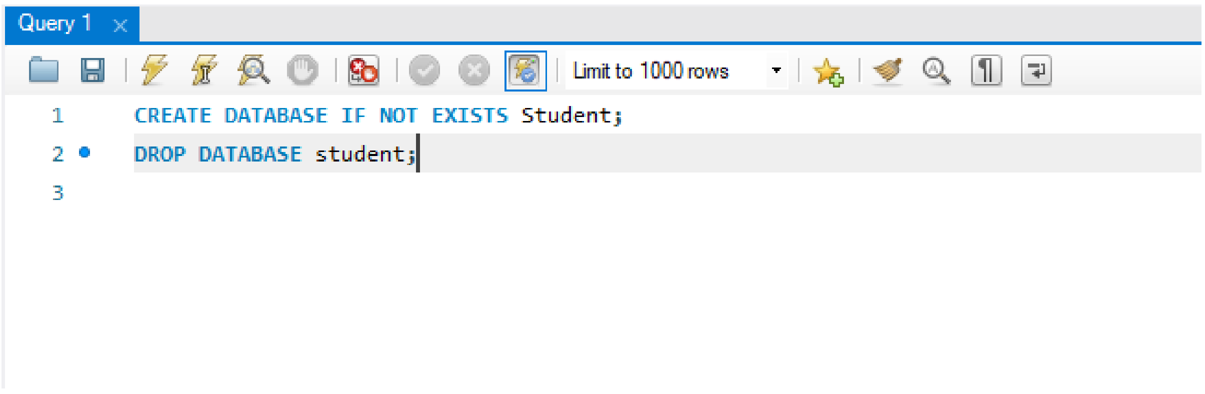
```
```

Create Database.



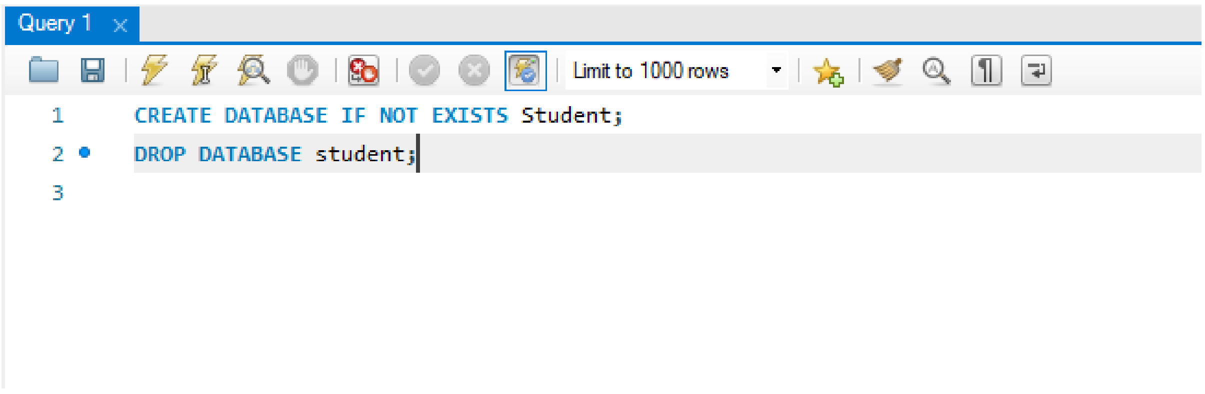
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

Create Table



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

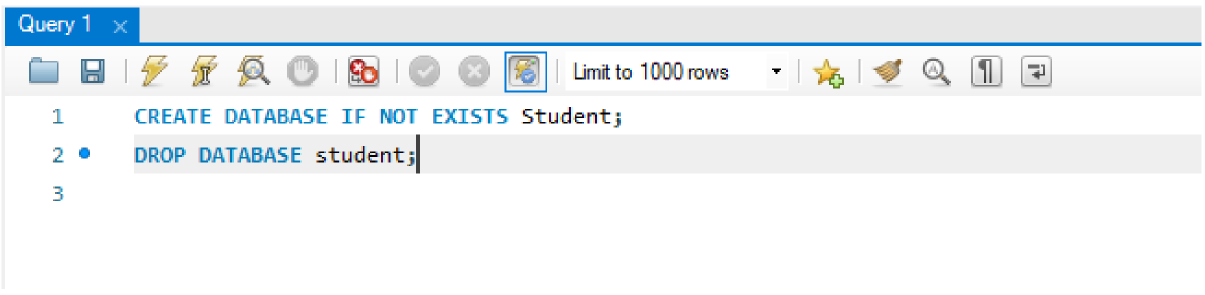
Drop Table



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

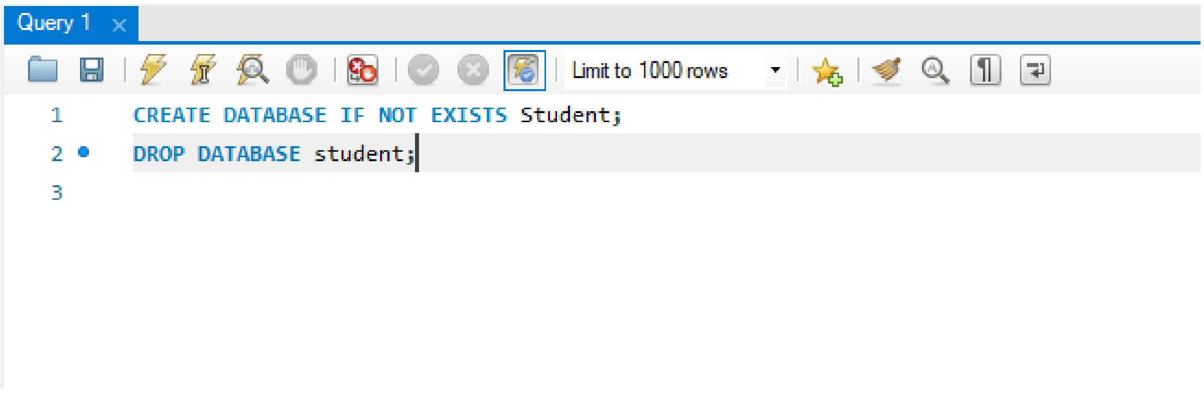
SELECT STATEMENTS

SELECT ALL



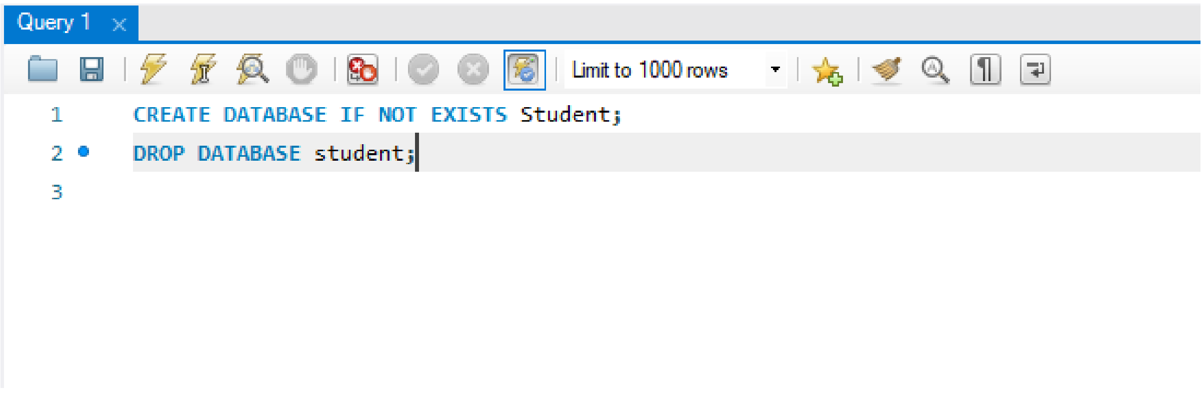
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT Particular Columns



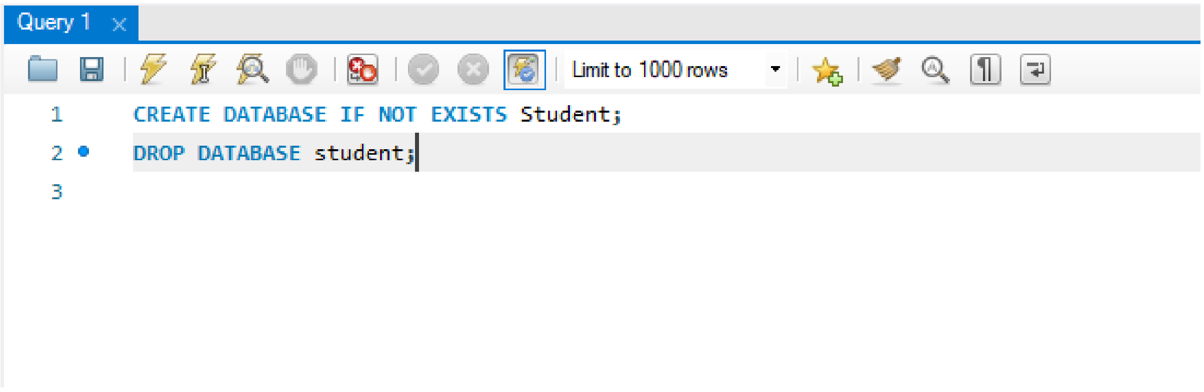
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT with WHERE STATEMENT



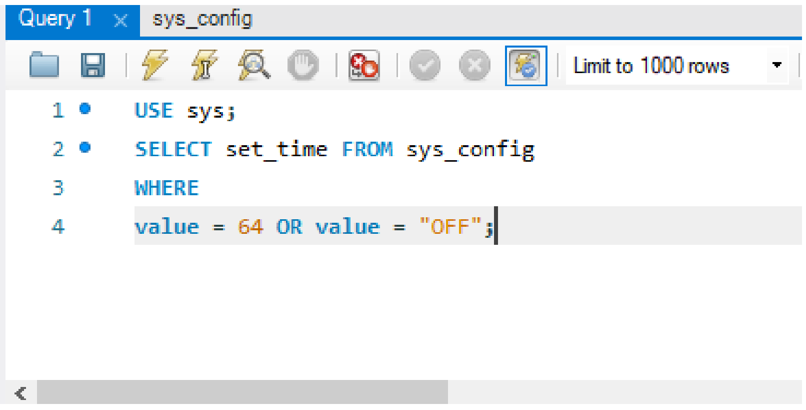
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT statement with AND operator

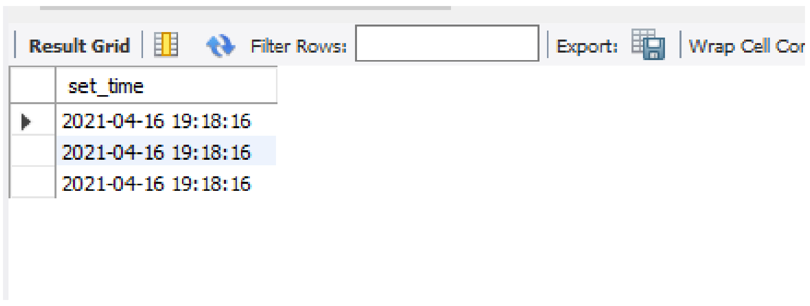


```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT statement with OR operator



```
USE sys;
SELECT set_time FROM sys_config
WHERE
value = 64 OR value = "OFF";
```



Result Grid | Filter Rows: | Export: | Wrap Cell Con

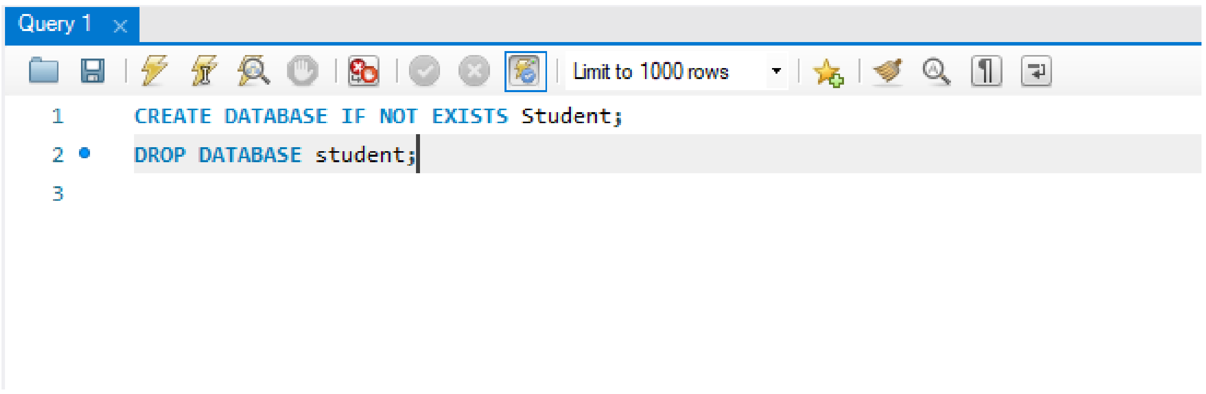
```
set_time
2021-04-16 19:18:16
2021-04-16 19:18:16
2021-04-16 19:18:16
```

AND :- Condition set on different column

OR :-Condition set on Same column.

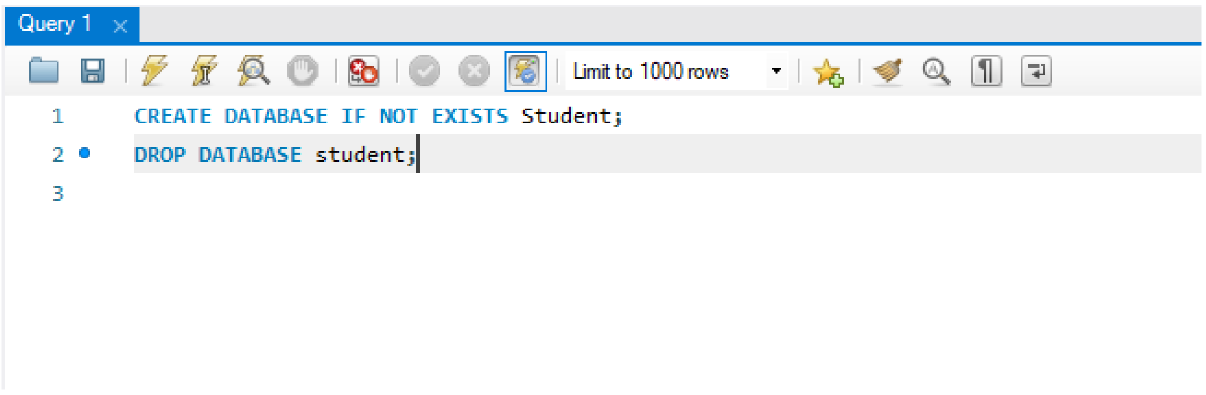
SELECT statement with IN and NOT IN operator

IN Operator



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

NOT IN Operator



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT statement with LIKE and NOT LIKE operator

On the basis of pattern we want to fetch the data from the database,in that case we will use like and Not like operator.

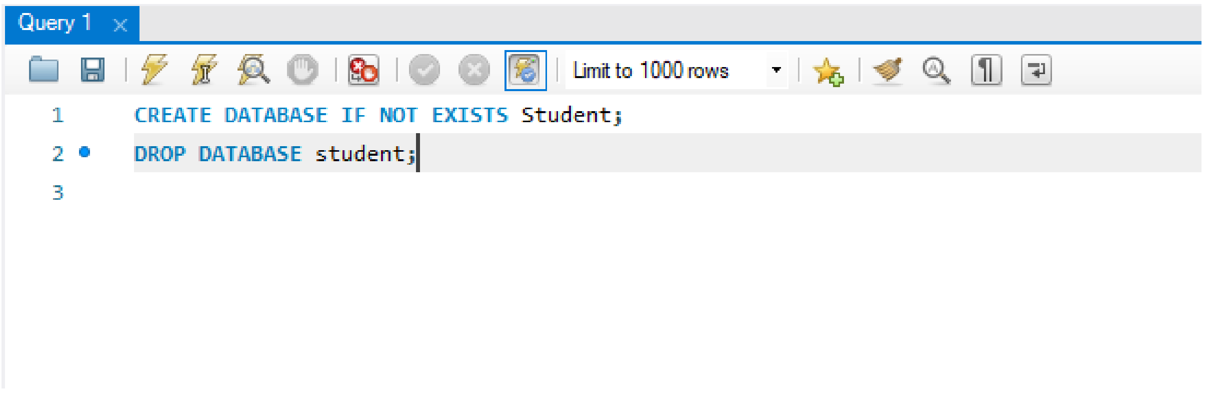
There is some wildcards we use generaly that is `%` and `_`

In the beginning of `%` any string we will use that will counted as string begining of that particular column and accordingly it will fetch the data voice versa.If we use any string inside two percentage sign,that is means it will fetch the data which matches the string pattern or string which is avaliable inside the percentage symbol.

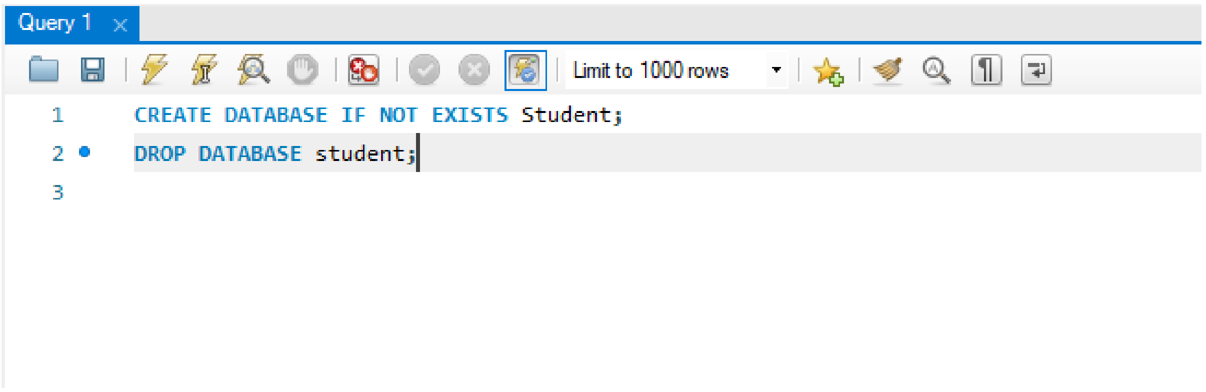
In underscore we will fecth the data on basis of position of the string.

`__mit` after the two position we will get the number as mit,so by using this similarity we can fetch the data.

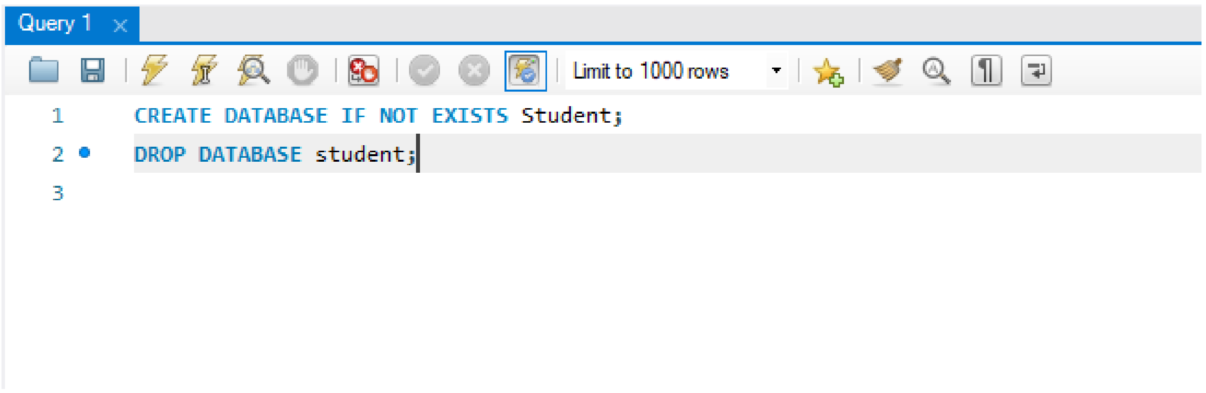
LIKE operator.



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

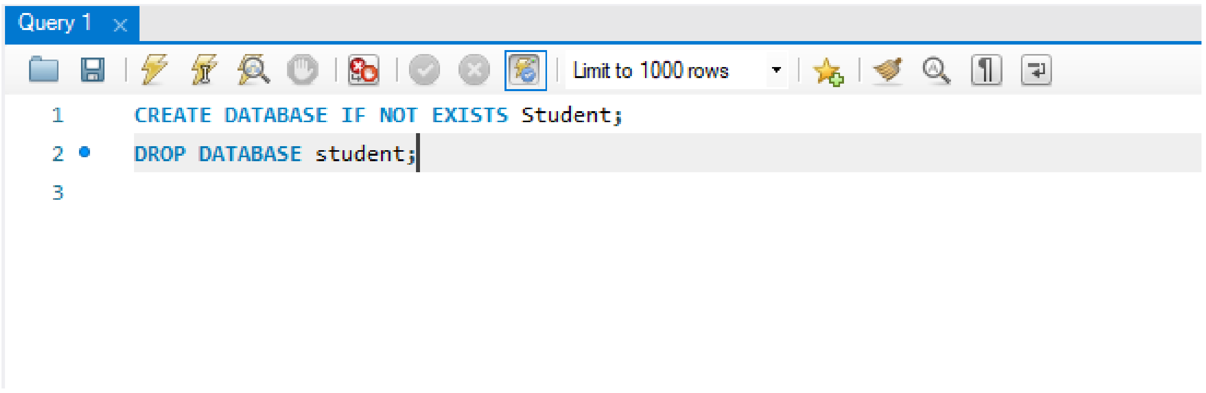


```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

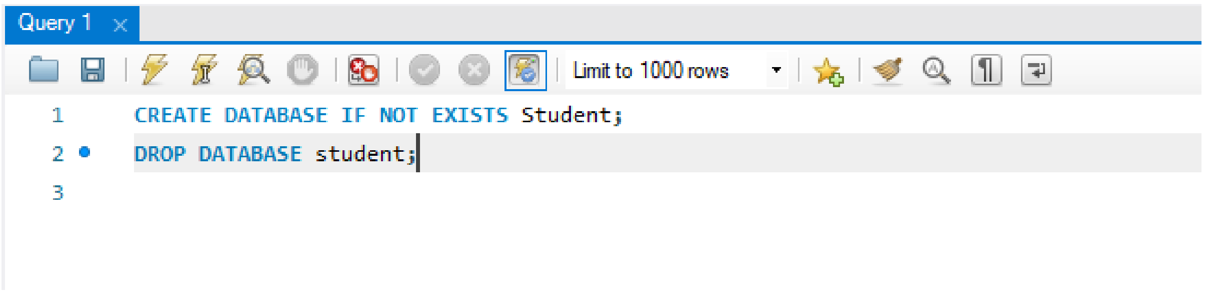


```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

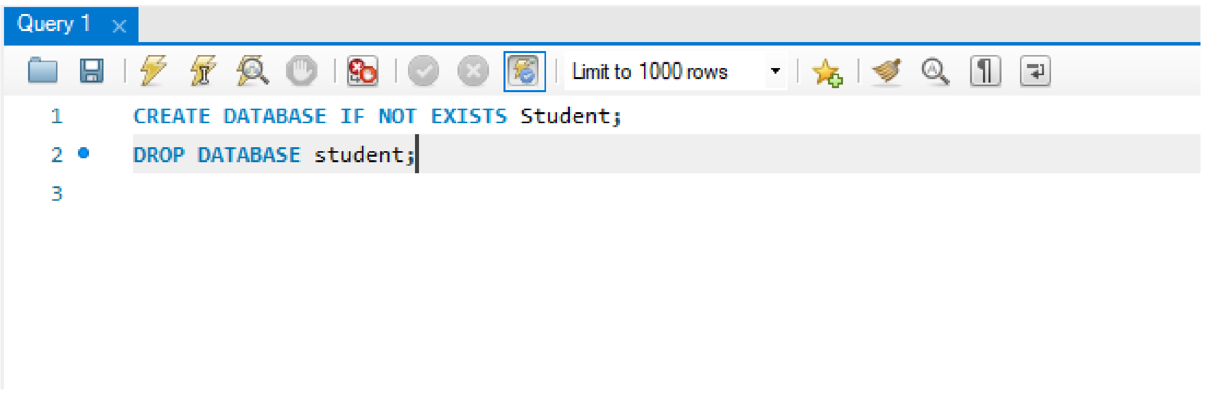
NOT LIKE Operator



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

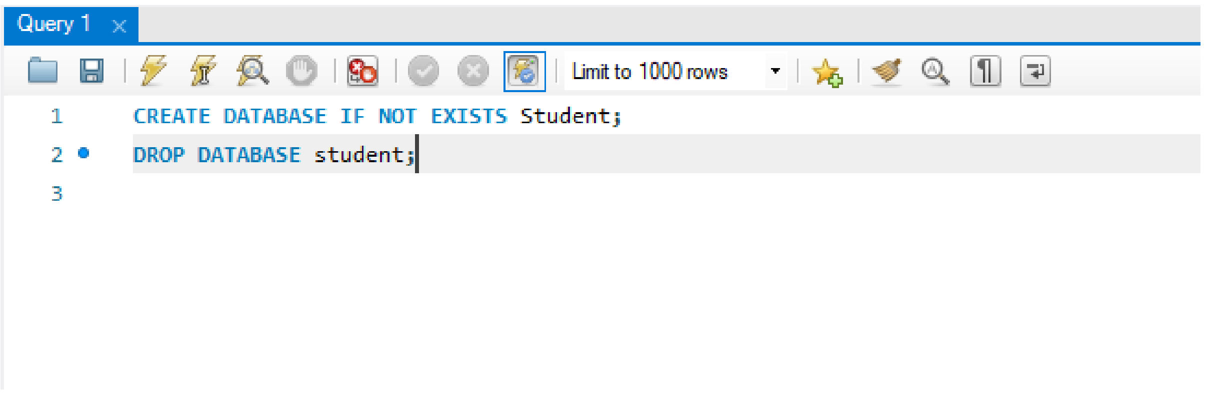


```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```



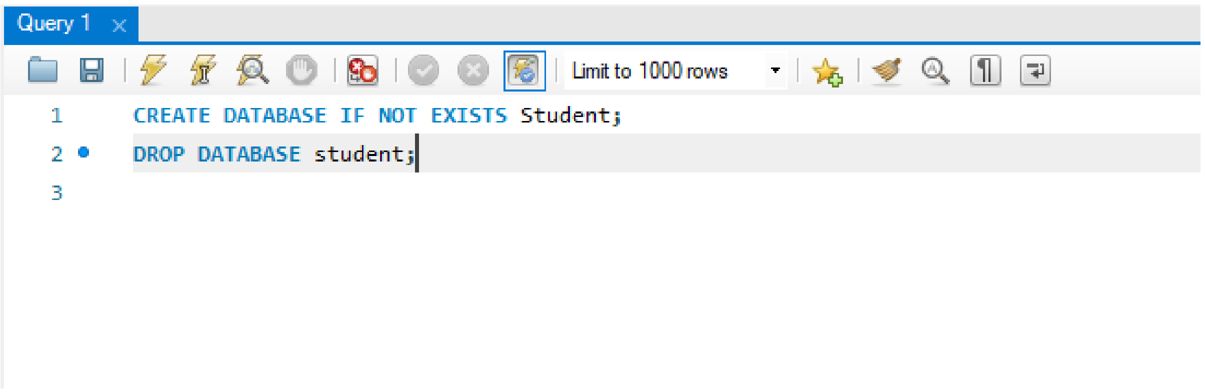
```sql
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT statement with BETWEEN AND



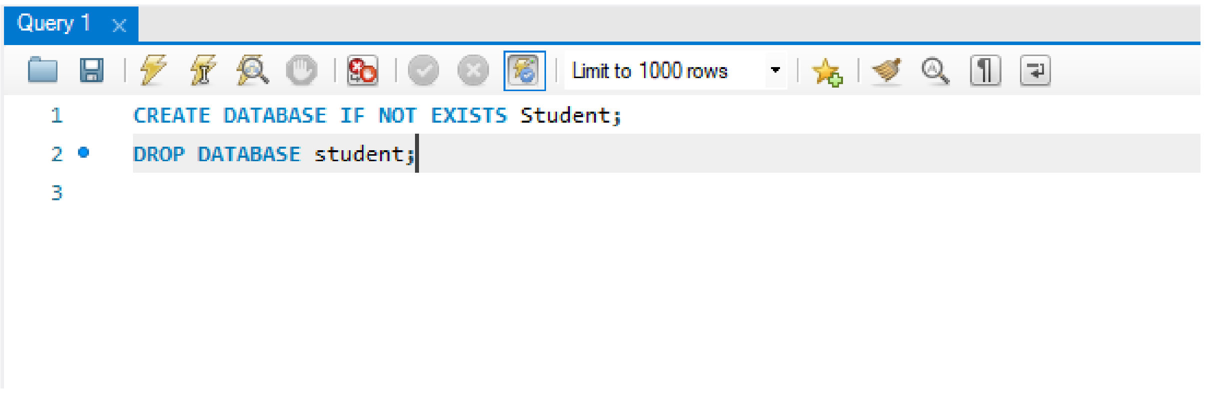
```sql
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT statement with NOT NULL



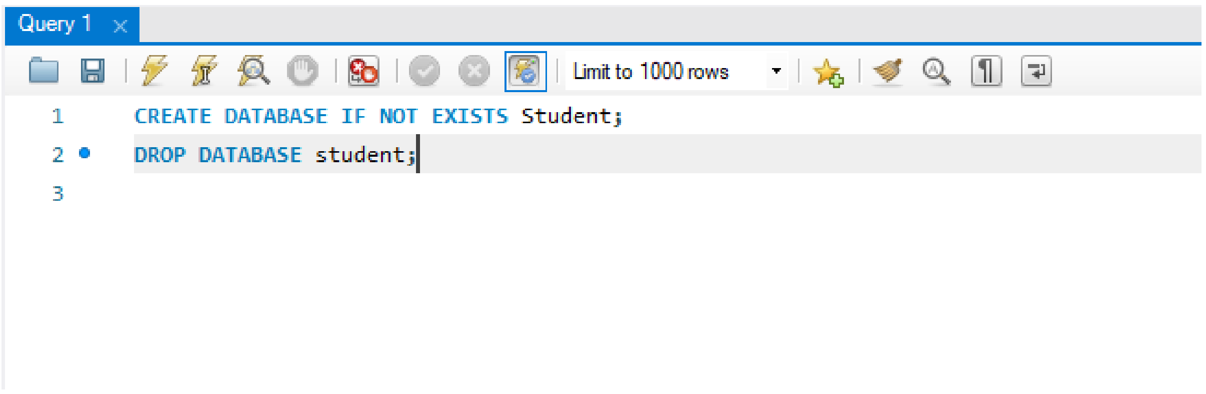
```sql
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT statement with NULL



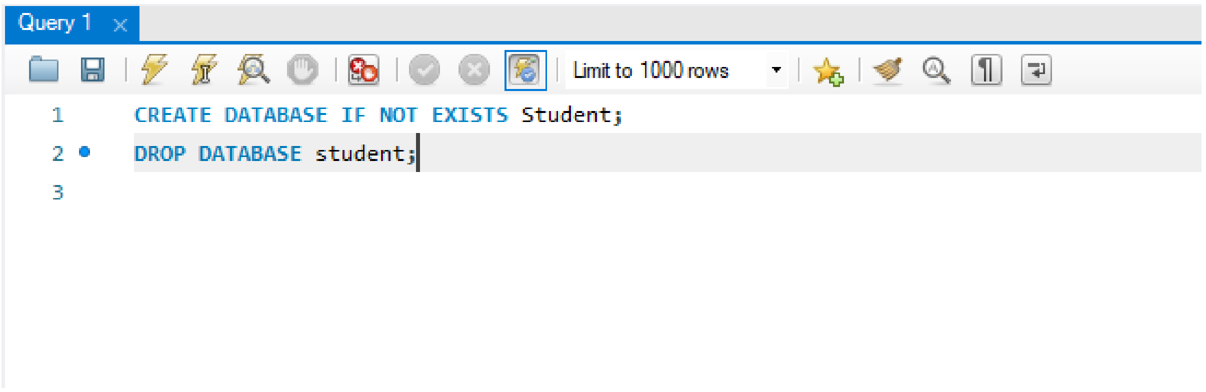
```sql
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

Other comparision Operator


Query 1

Limit to 1000 rows

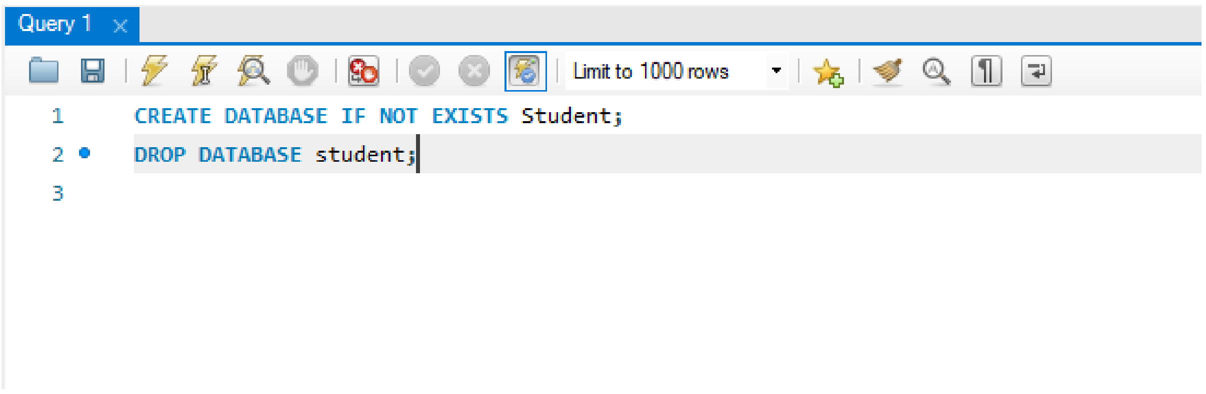
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```


Query 1

Limit to 1000 rows

```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

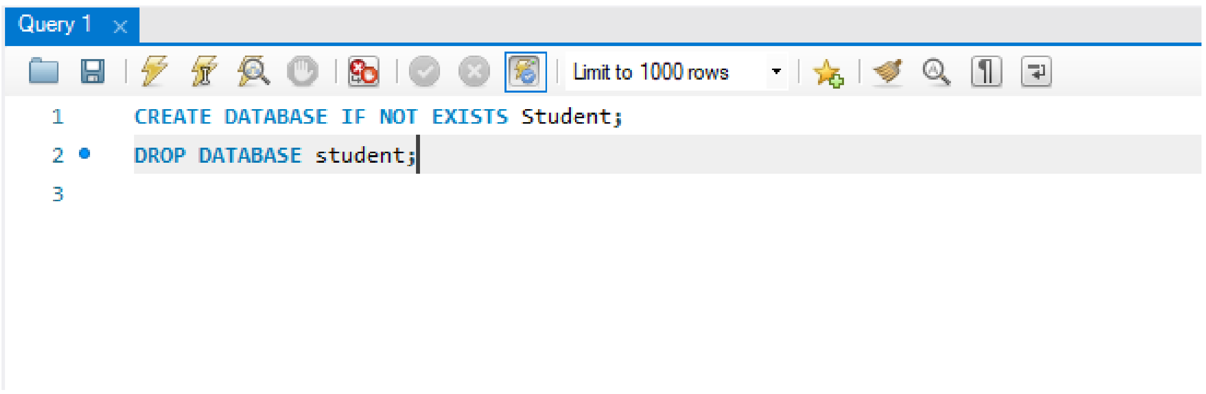
SELECT with DISTICT operator.


Query 1

Limit to 1000 rows

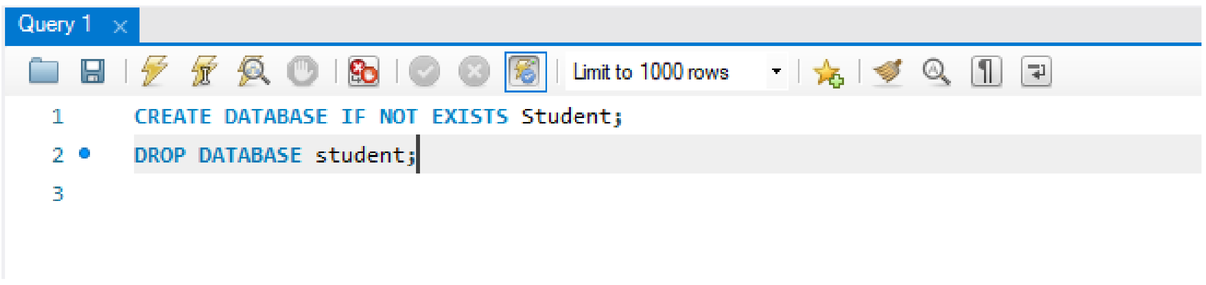
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

Aggregate Functions Introduction.


Query 1

Limit to 1000 rows

```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

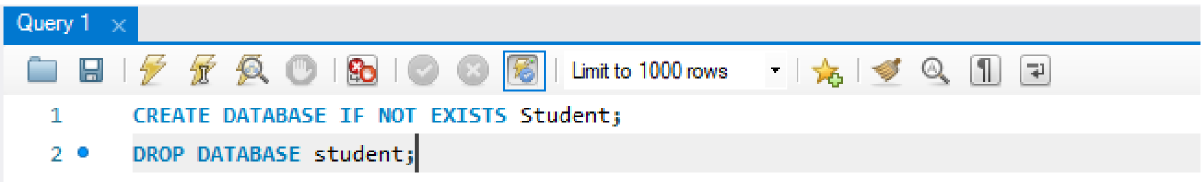

Query 1

Limit to 1000 rows

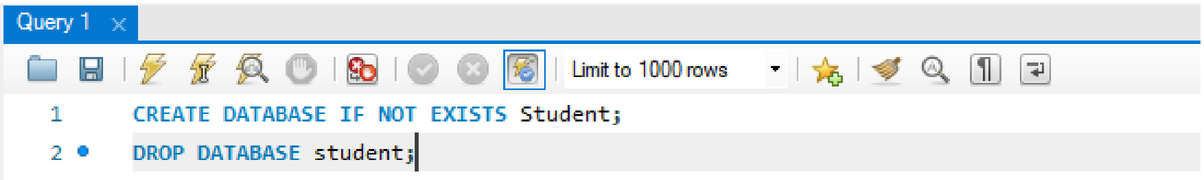
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

The aggregated function ignore the null values untill told not to.

SELECT statement with ORDER BY Statement.

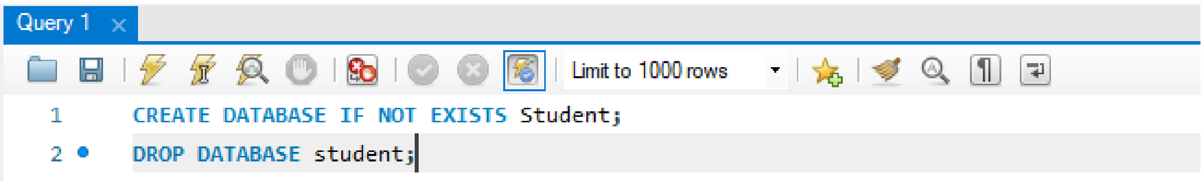


```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT statement with Alliases (AS)

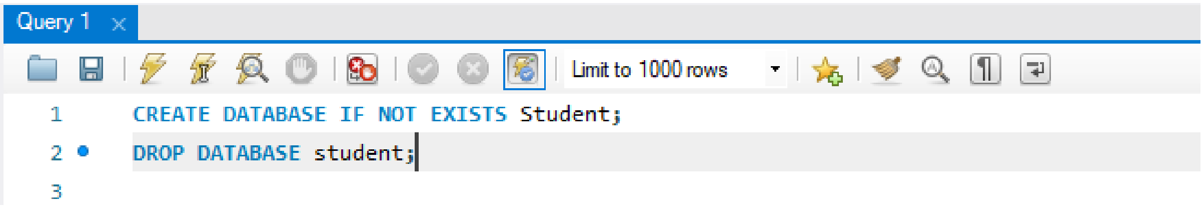


```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT STATEMENT with HAVING Clause.

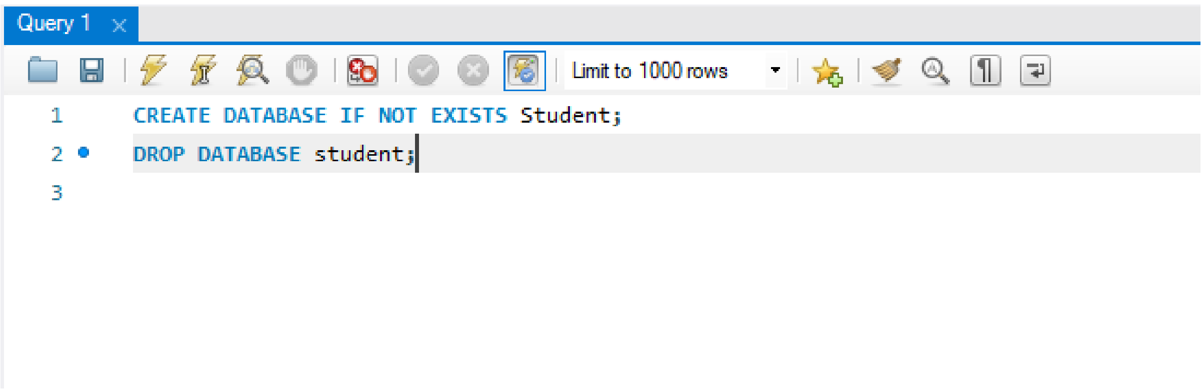
Having help us to refined output from the records that do not satisfy certain condition.It is frequently implemented with GROUP BY statement.

Code Structure :-



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

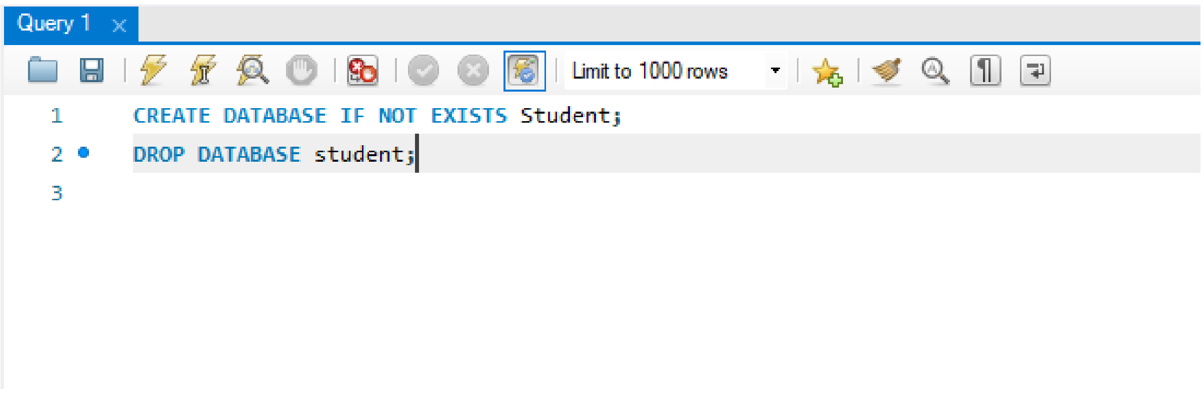
Having like WHERE but applied on top of GROUP BY block.



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

SELECT statement with LIMIT parameter

It returns the 5 rows from the table which we want to fecth from MySQL database.

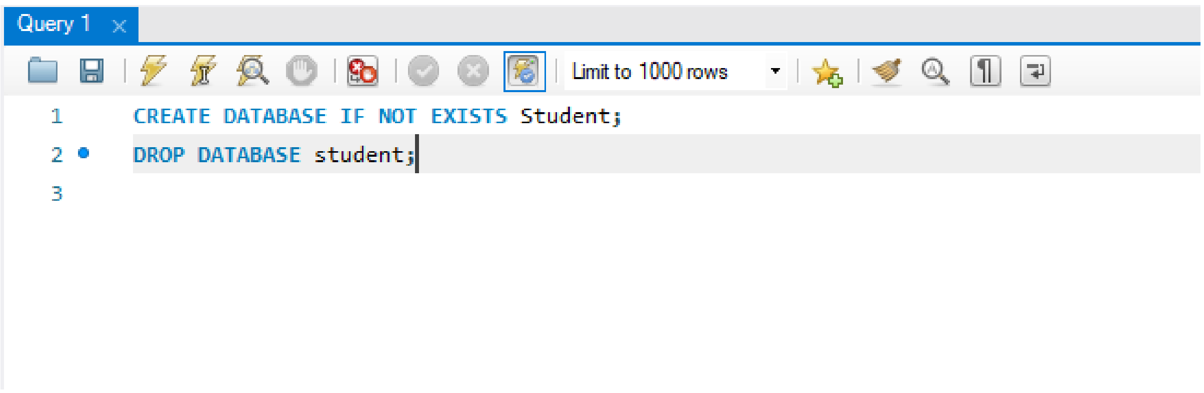


```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

CRUD Operator

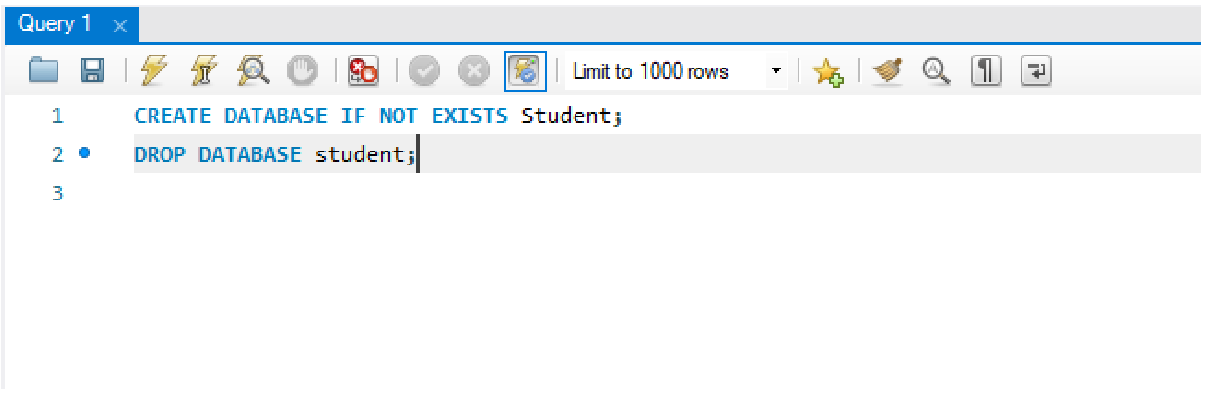
INSERT Statement.

In the first we have decide in which column we want to insert the data and then use the value keyword so that we can add the data.



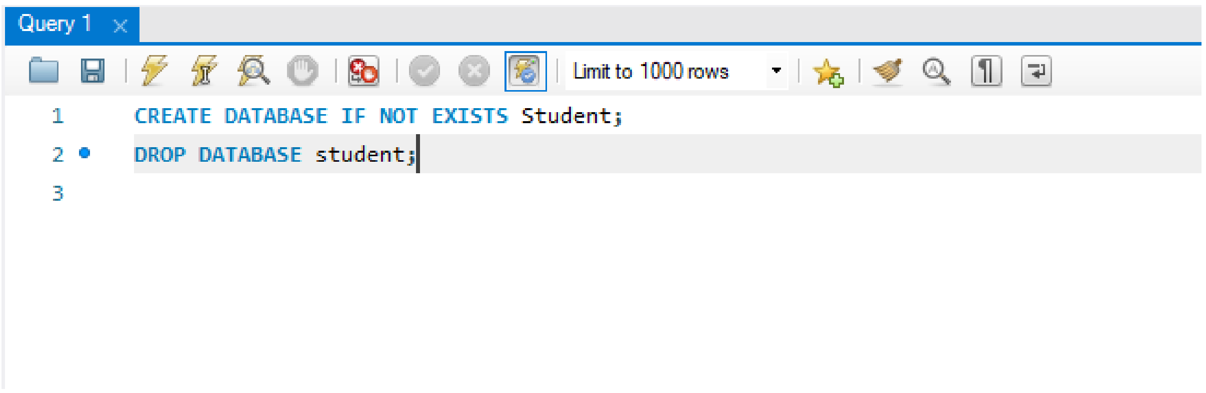
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

UPDATE Statement



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

DELETE Statement



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

READ Statement

This is nothing but the SELECT statement

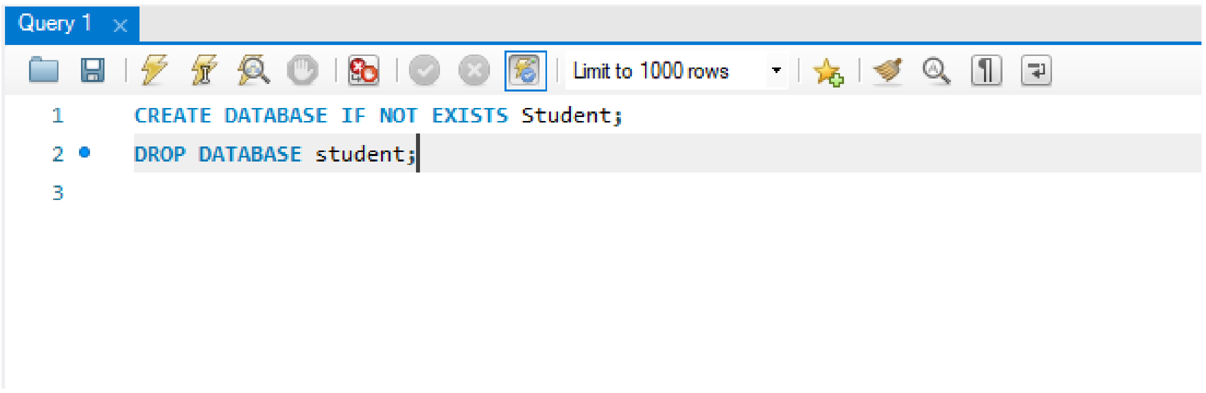
Commit and Roll Back

COMMIIT :- The COMMIT statement commits the database changes that were made during the current transaction, making the changes permanent.

ROLLBACK:- statement backs out, or cancels, the database changes that are made by the current transaction and restores changed data to the state before the transaction began.

COMMIT permanently saves the changes made by current transaction. ROLLBACK undo the changes made by current transaction. Transaction can not undo changes after COMMIT execution. Transaction reaches its previous state after ROLLBACK.

Below figure showing we do the roll back many time but it reference to last excution of commit



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

DROP Vs TRUNCATE Vs DELETE

DROP :-

In the DROP command we will loose everything with values and structure of the table.You won't able to rollback to initial state,or the LAST COMMIT Statement

Truncate :-

In the trucate we can remove the values inside the table but the structure of that table remain unchanag.In thruncate Auto Incremenet will be reset.

DELETE:-

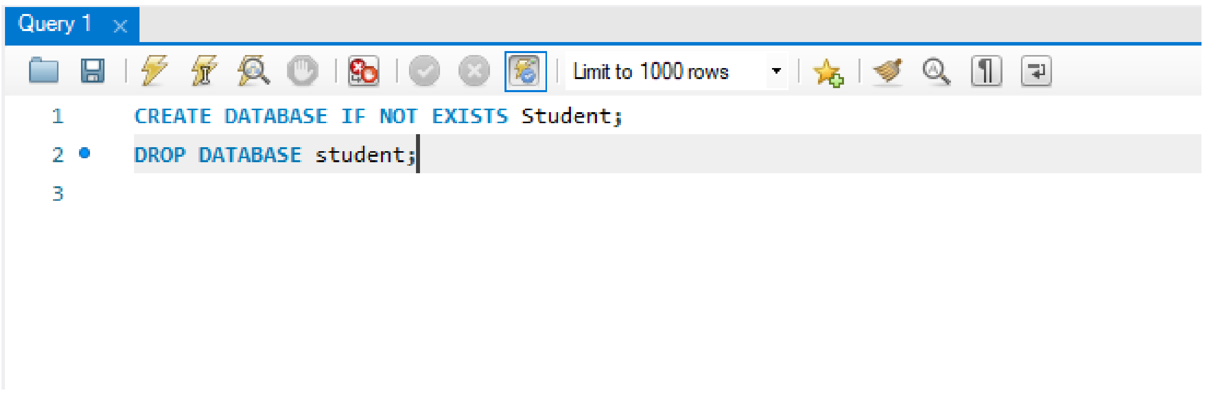
It is helping us to DELETE the records row by row.

In the DELETE AND TRUNCATE we have where condition.But in the case execution Truncate excute fast code than th DELETE.

ALTER Statement

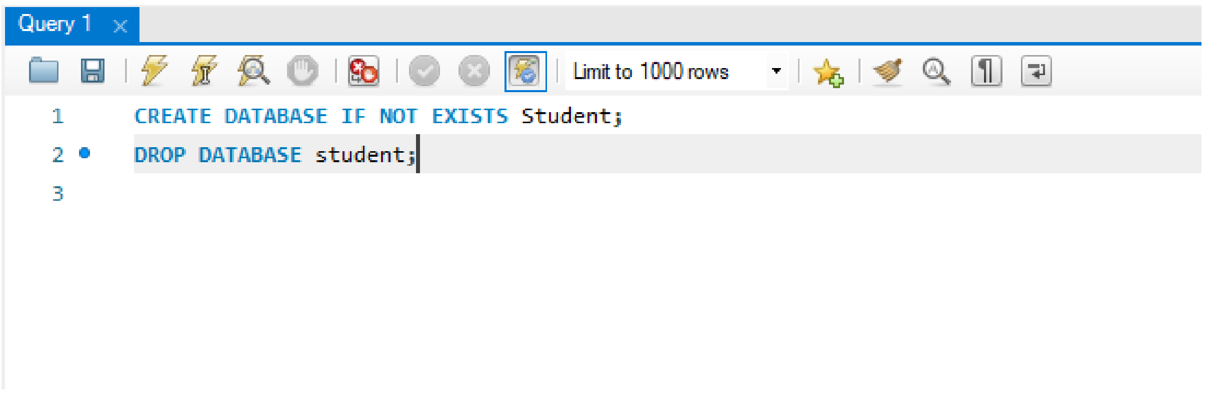
- **The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.**
- **The ALTER TABLE statement is also used to add and drop various constraints on an existing table.**

DROP Table



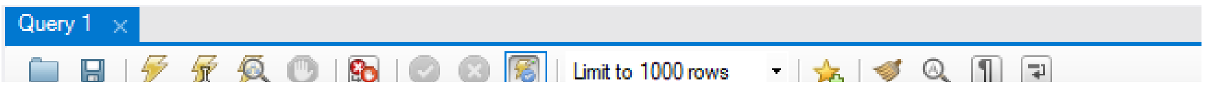
```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

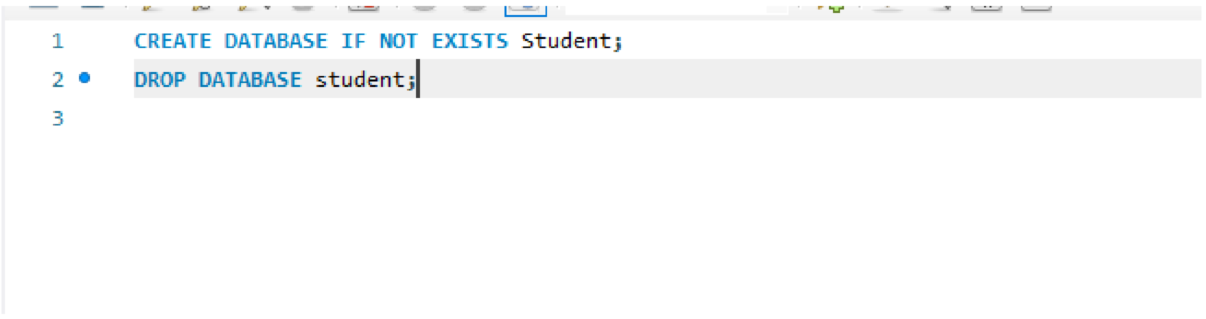
MODIFY Table



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

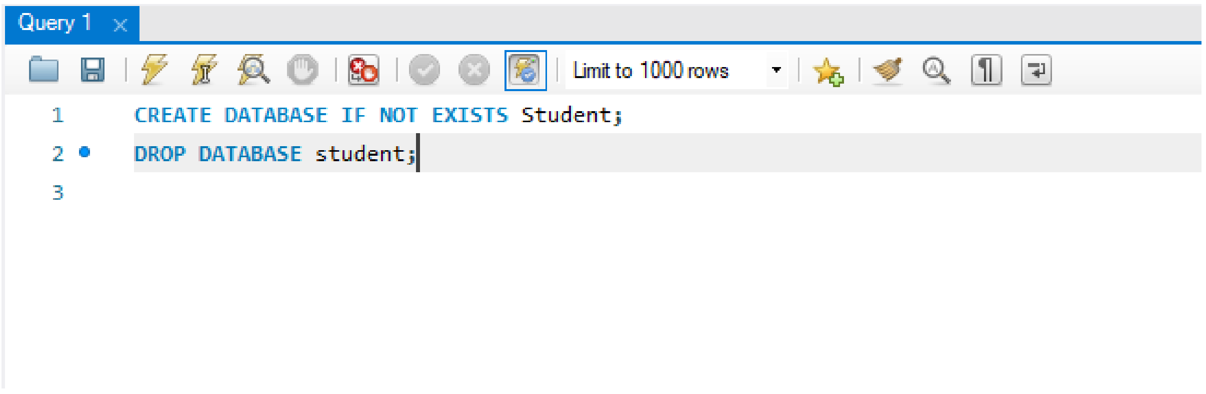
RENAME Table





```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

ADD Column



```
CREATE DATABASE IF NOT EXISTS Student;
DROP DATABASE student;
```

In [ ]:

Primary Key

The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

Foreign Key

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

A foreign key (FK) is a column or combination of columns that is used to establish and enforce a link between the data in two tables to control the data that can be stored in the foreign key table

Default

The DEFAULT constraint is used to set a default value for a column. The default value will be added to all new records, if no other value is specified.

Not Null

The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

Auto Increment.

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

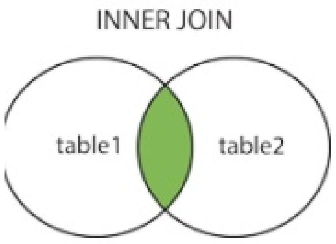
In order to get the information from more than one table at time then we have to join the tables on the certain conditions like below way.

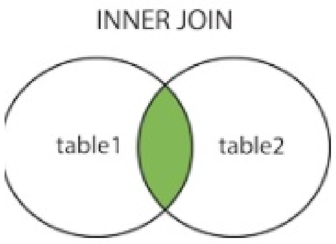
There are several kind of MySQL joins avalible which are given below :-

- **1.Inner join or join**
- **2.Outer join or Union join**
- **3.LET join**
- **4.RIGHT join**
- **5.Cross Join**
- **6.SELF join**

Inner Join Or Join

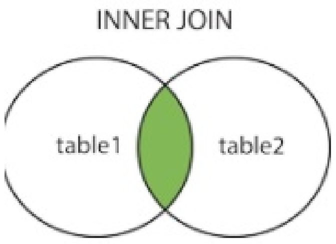
It is the on of the table joining method which is helping to join the table on the matching data between two or more table.



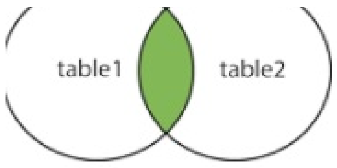


Another way of Inner join query writing style

By using the below way of joining we can specificaly getting data from the table,otherwise if we are going to fetch the data which is avalible inside the both table then SQL query will get confused and throw the error like ambiquity error.

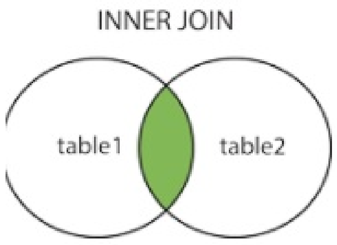


Three Tables INNER JOIN



Many times the data in joining is getting duplicated so in order to avoid the duplication of the data we have to use the GROUP BY.

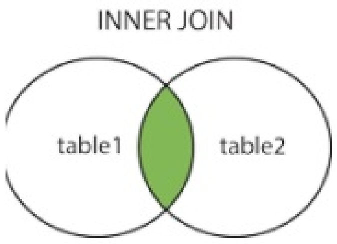
You can not allow yourself to assume there are no duplicate rows in your data.

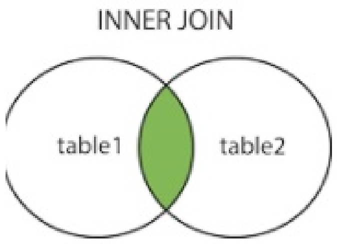


- **LEFT TABLE :- Contains the FROM Statement**
- **RIGHT TABLE :- Contains the Join On statement.**

LEFT Join

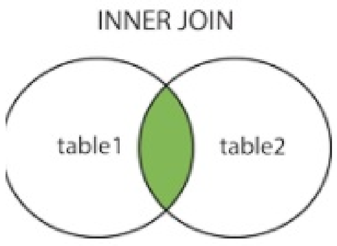
It is the join which can include the total data avaliable inside the LEFT side table and matching data Inbetween the LEFT and RIGHT table and Exclude the unmatched RIGHT table data.

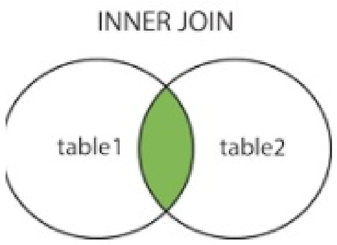




RIGHT Join

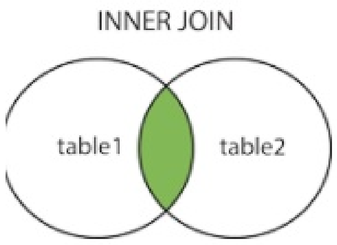
It is the join which is helping us to get the matching data from Both table and data which is avaliable inside the RIGHT Table and Excluding the LEFT SIDE unmatched table.





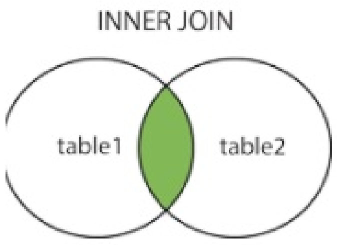
We get the data by using the WHERE clause while the JOIN table but it is good for very low data and if use for big data then it will be a time consuming for us.Thats why we prefer the Join Clause Instead of WHERE Clause.

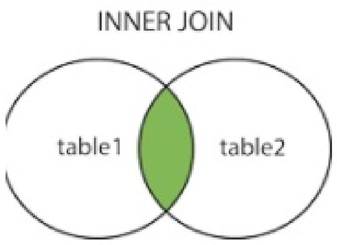
Where Clause With Inner Join.

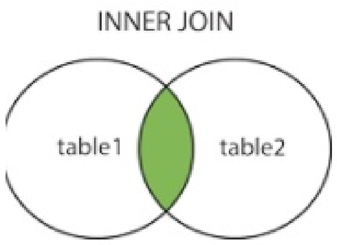


Cross Join

- **Cross join will the values from the certain table and connect them with all the values from the tables we want to join it with.**
- **Typically connect with matching values.**
- **A cartesian product of values two or more sets.**
- **Particulary useful when table in database are not well connected.**
- **The CROSS JOIN is used to generate a paired combination of each row of the first table with each row of the second table. This join type is also known as cartesian join.**



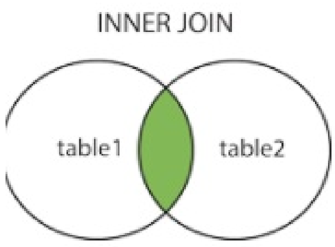


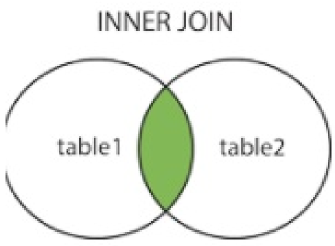


The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN.This kind of result is called as Cartesian Product.

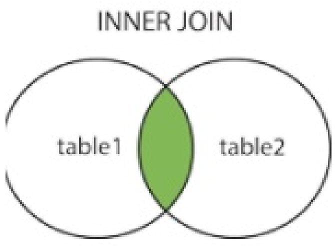
If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

An alternative way of achieving the same result is to use column names separated by commas after SELECT and mentioning the table names involved, after a FROM clause.

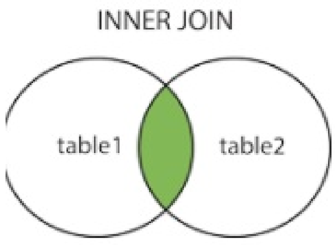




Aggrigate Function With Join Tables.

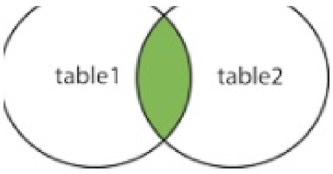


Outer Join Or UNION Join.



Union All

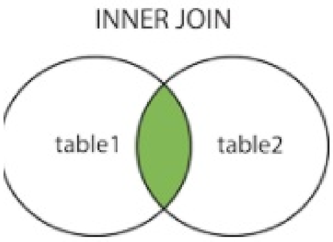
It is combine the few select statements in single output,you can think of it as a tool that allows to you to unify tables. The SQL UNION ALL operator is used to combine the result sets of 2 or more SELECT statements. It does not remove duplicate rows between the various SELECT statements (all rows are returned). Each SELECT statement within the UNION ALL must have the same number of fields in the result sets with similar data types



The UNION ALL command combines the result set of two or more SELECT statements (allows duplicate values).

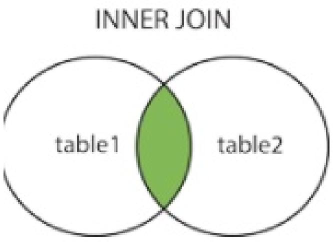
Union

UNION display only the distinct values in the output.



SELF Join

It is applied when the table must join itself,if we like to combine ceratin rows of the table with other rows of the same table,you need self-join



In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

About Sub-Queries.

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow –

- **Subqueries must be enclosed within parentheses.**
- **A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.**
- **An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.**
- **Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.**
- **The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.**
- **A subquery cannot be immediately enclosed in a set function.**
- **The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.**

Rules to Use Subqueries in SQL

We need to follow some rules while writing SQL Subqueries. We will discuss the rules below:

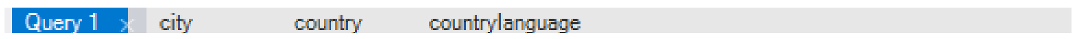
- **Subqueries need to be enclosed in the Where clause and can be used with Insert, Update, Delete, and Select statements.**
- **We can use comparison operators for example: <, >, > =, < =, !=, IN , Between for the subqueries.**
- **The subquery is always executed first and then the main query.**
- **Subquery should be enclosed within parentheses.**
- **Subqueries are always to the right of the comparison operators.**
- **We can't use Order By clause in the subquery; instead, we can use the Group By clause.**
- **We should use single-row operators with single-row subqueries and vice versa.**
- **We can't use Between clause with a subquery, but we can use Between in a subquery.**

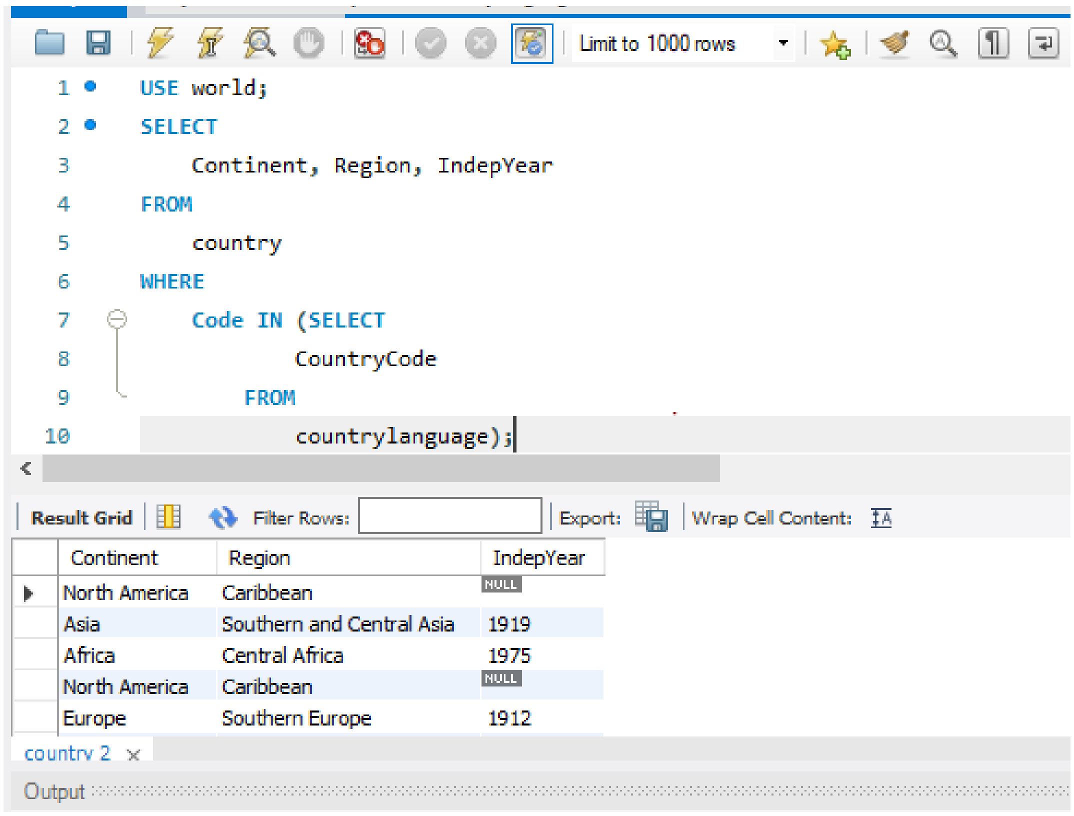
SQL subqueries or nested queries are SQL statements where we need the results from our database after using multiple filters.A subquery is put to restrict the data pool for the main query i.e., the inner query gives us the data which is the pool for the main query.

Types Of Subqueries

- **Single Row Subquery. Returns zero or one row in results.**
- **Multiple Row Subquery. Returns one or more rows in results.**
- **Multiple Column Subqueries. Returns one or more columns.**
- **Correlated Subqueries. Returns one or more columns according to the main or the outer query, thus called a correlated subquery.**
- **Nested Subqueries.We have queries within a query(inner and outer query).**

Nested Queries with Where,IN,ALL,ANY Clause





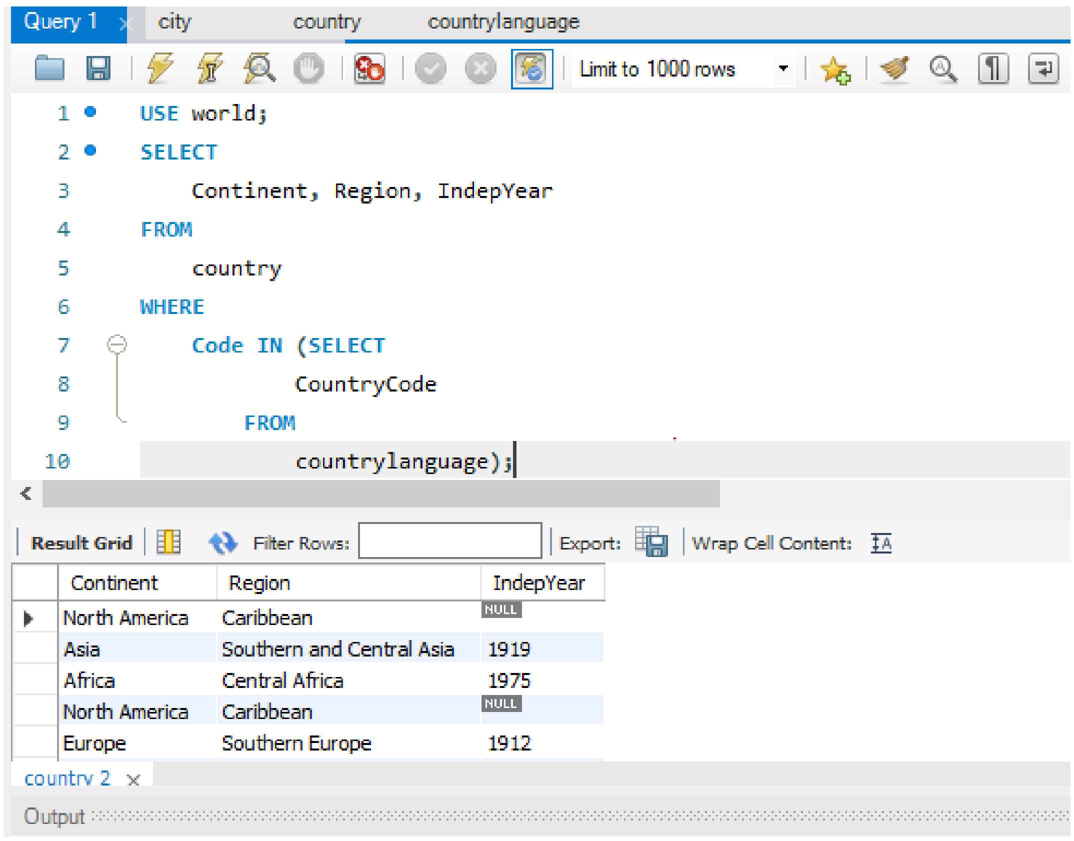
```sql
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

In SQL engine starts by running the inner query and then it uses its returned output,which is intermediate,to execute the outer query.

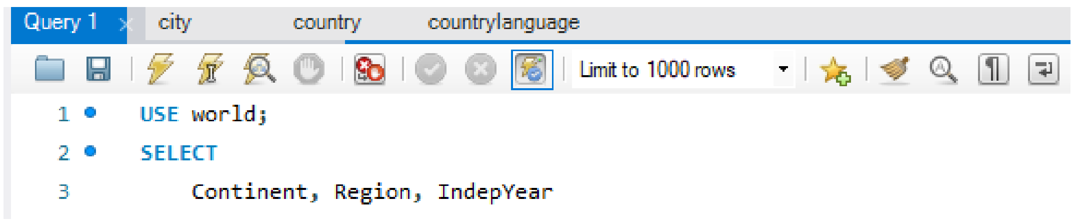
Subqueries with EXISTS and NOT-EXISTS Statement

It returns the boolean output which is nothing but True and False and check is conducted row by row.

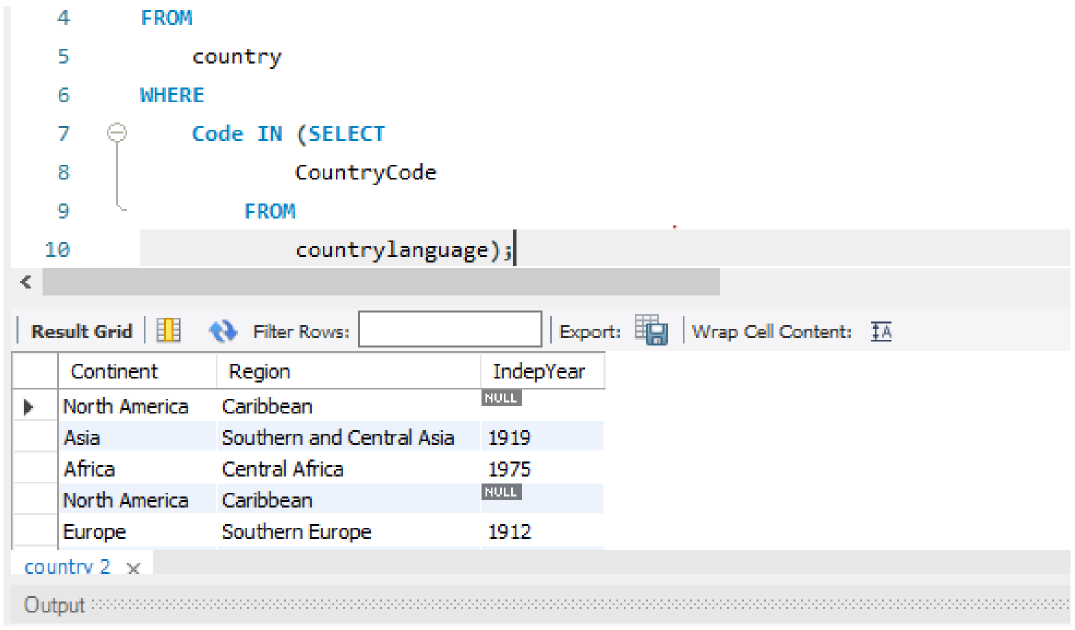


```sql
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912



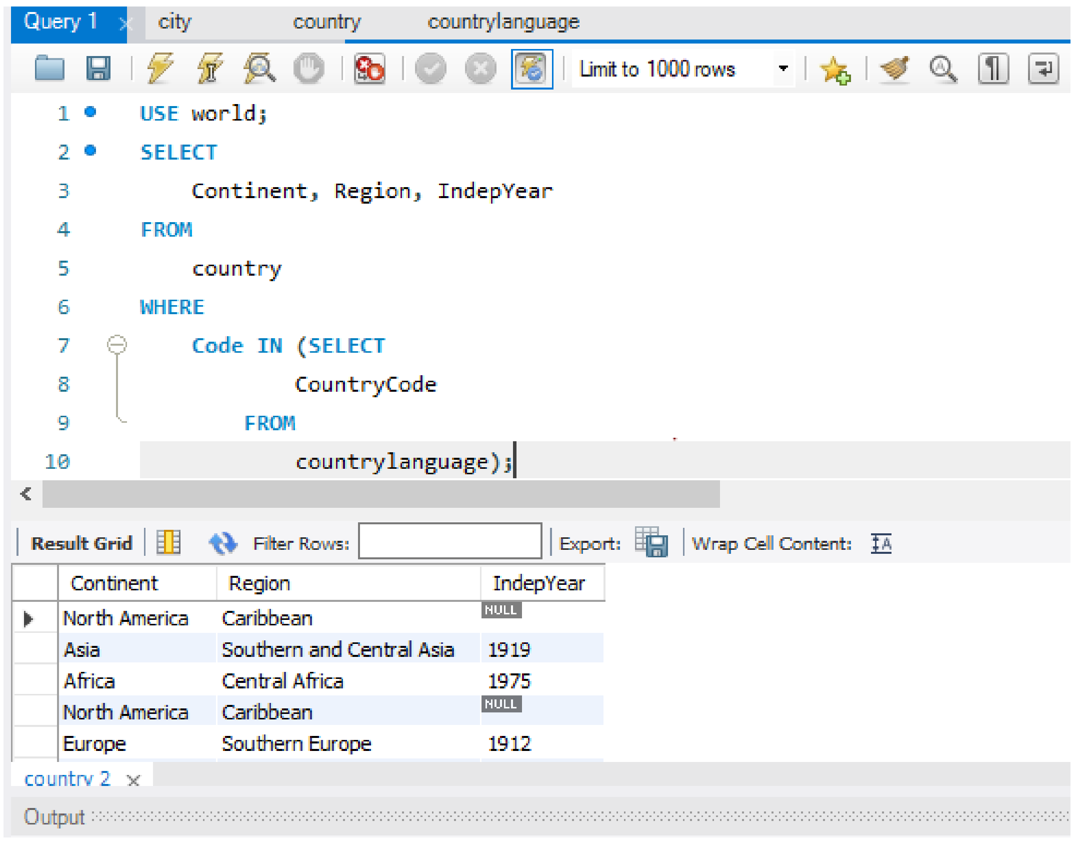
```sql
USE world;
SELECT
    Continent, Region, IndepYear
```



```
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

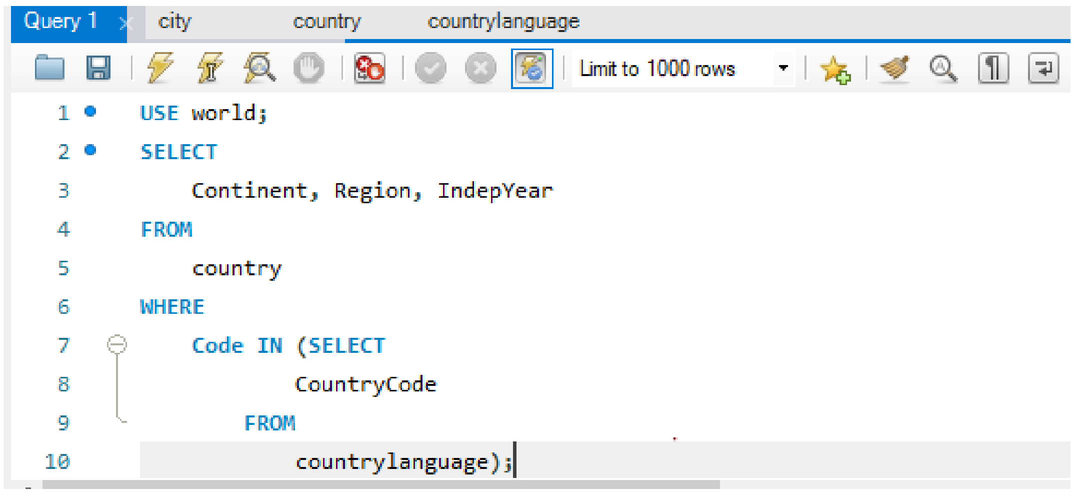
Always Use Order By Outside the Inner Query.



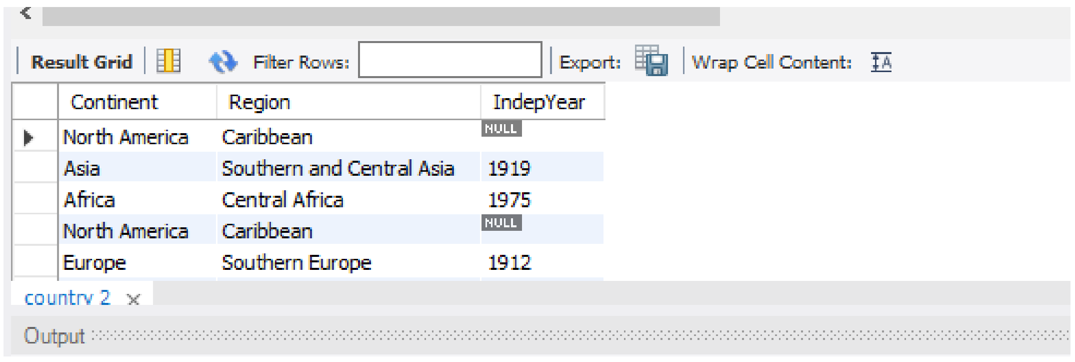
```
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

Subquery With SELECT FROM Statement.



```
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```



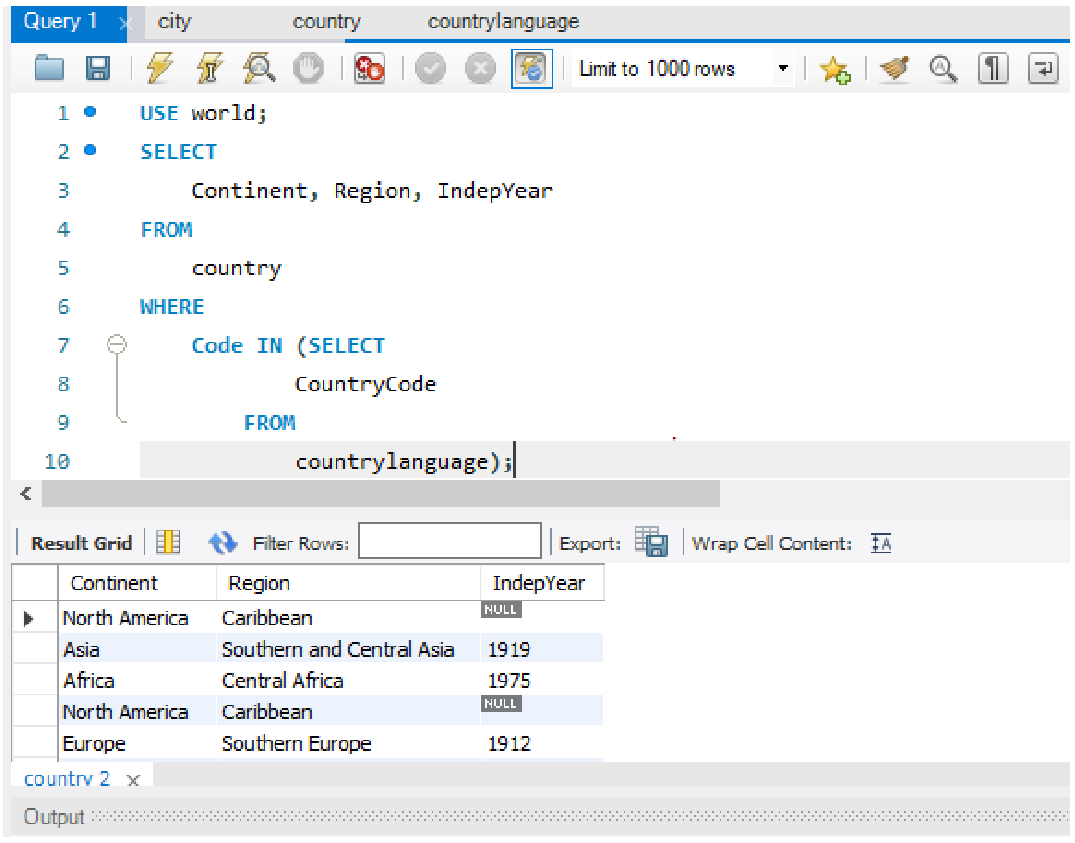
Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

Example about Types of Subqueries.

1. Single Row Sub-Quries.

In returns zero or only one row.

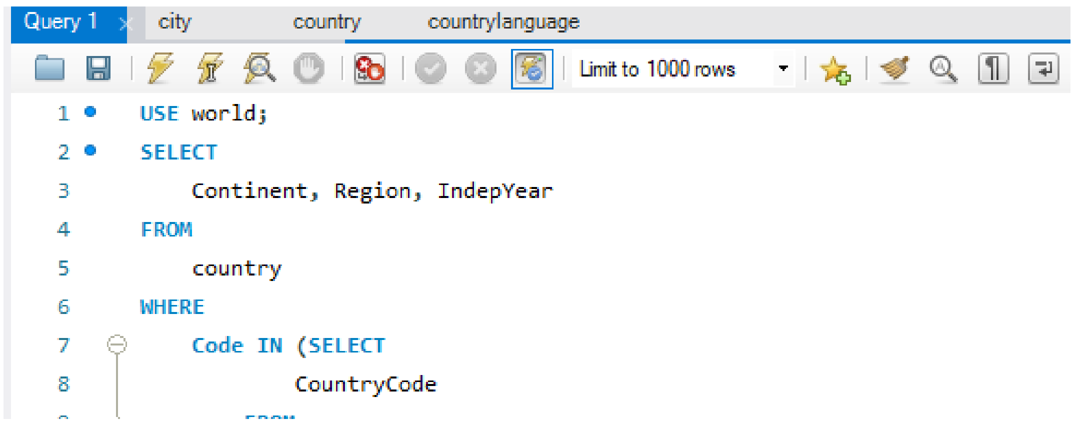
We use single row comparision operator.



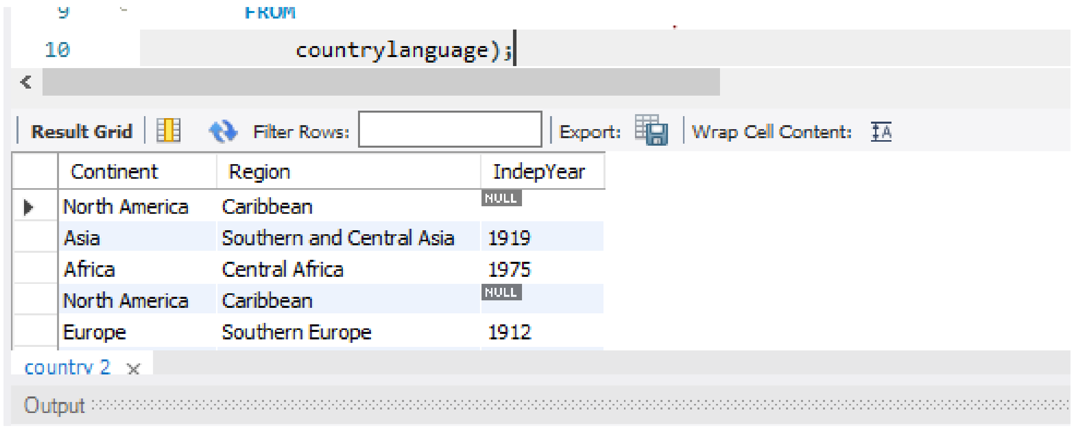
```
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

A single-row subquery is used when the outer query's results are based on a single, unknown value. Although this query type is formally called "single-row," the name implies that the query returns multiple columns-but only one row of results



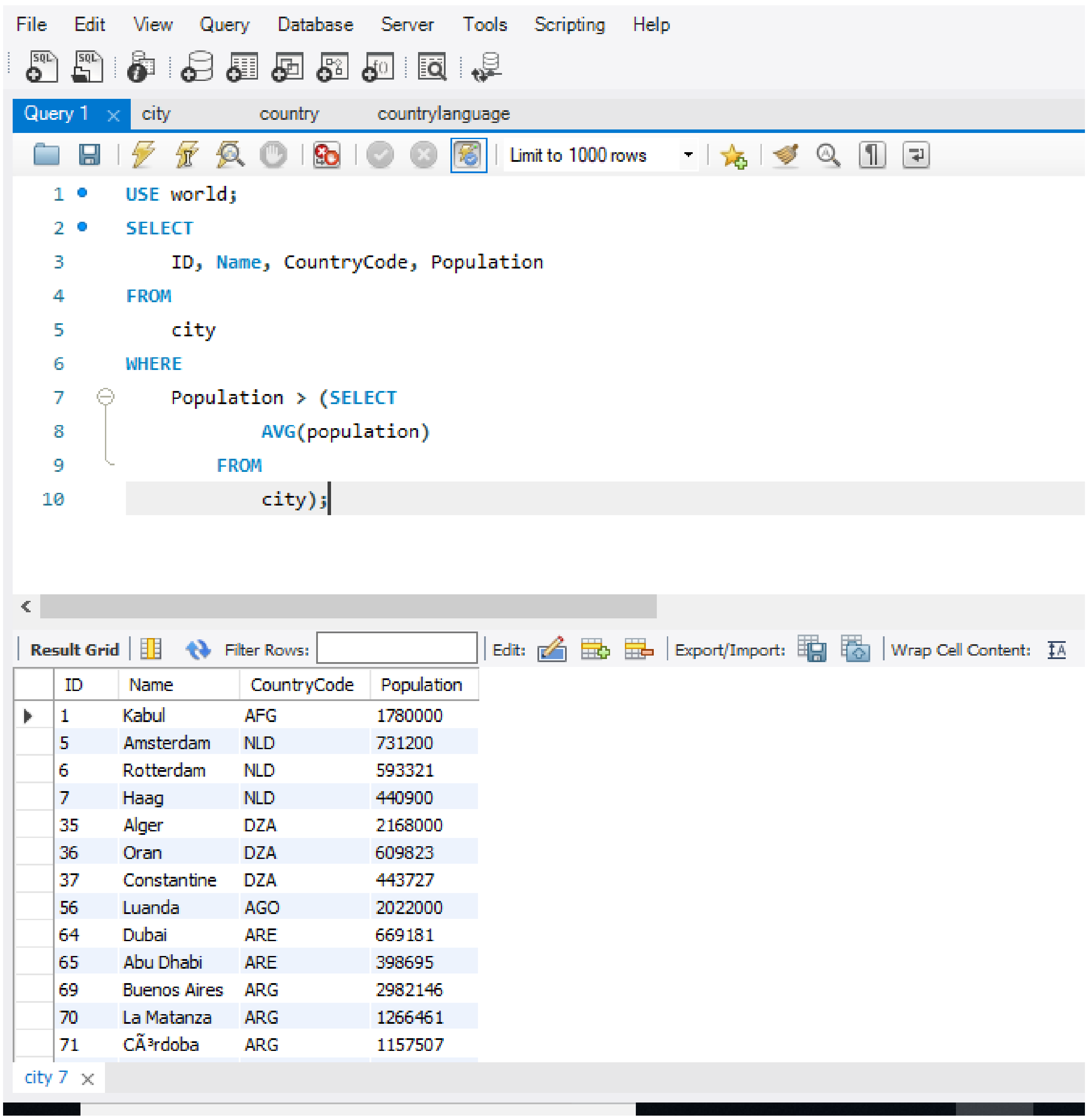
```
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
```



```
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

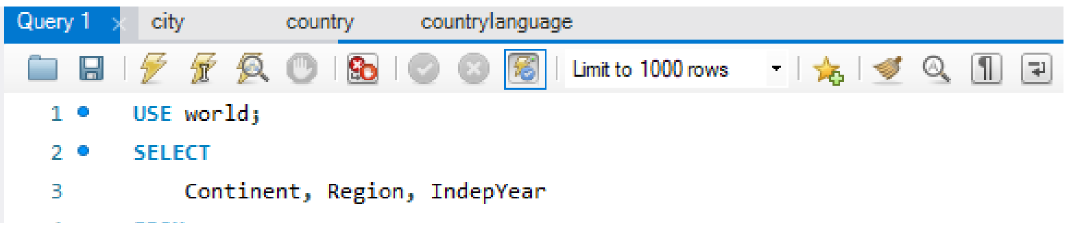
In the below way we can it returning whichever NAME,Countrycode,Population greater than average population then we are sigle row for mutiple column.



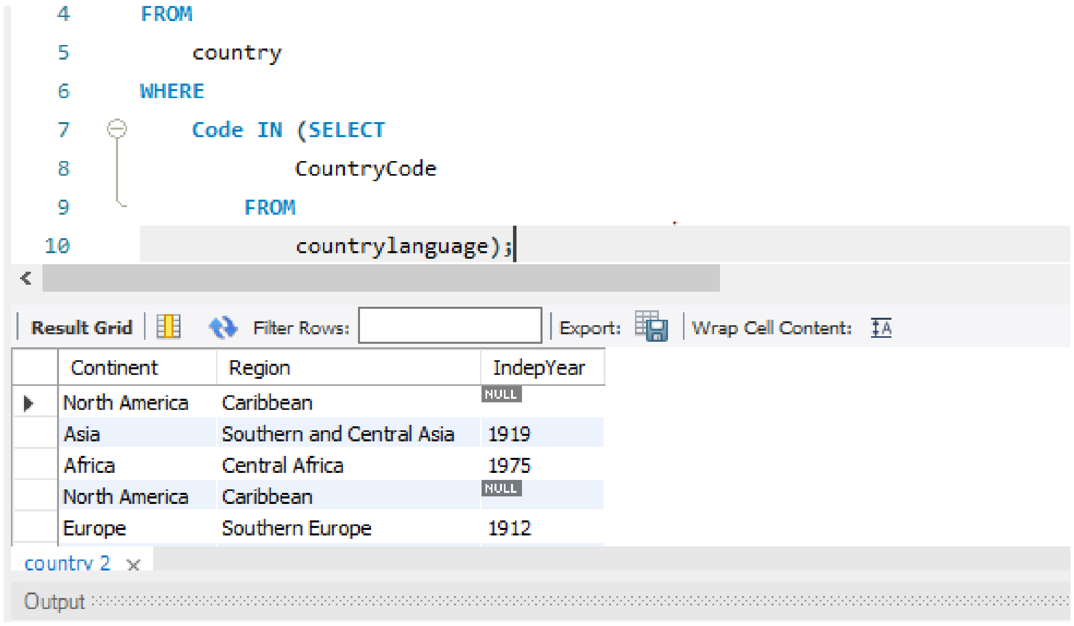
```
USE world;
SELECT
    ID, Name, CountryCode, Population
FROM
    city
WHERE
    Population > (SELECT
            AVG(population)
        FROM
            city);
```

ID	Name	CountryCode	Population
1	Kabul	AFG	1780000
5	Amsterdam	NLD	731200
6	Rotterdam	NLD	593321
7	Haag	NLD	440900
35	Alger	DZA	2168000
36	Oran	DZA	609823
37	Constantine	DZA	443727
56	Luanda	AGO	2022000
64	Dubai	ARE	669181
65	Abu Dhabi	ARE	398695
69	Buenos Aires	ARG	2982146
70	La Matanza	ARG	1266461
71	CÃ³rdoba	ARG	1157507

2. Multi-Row Sub-Quries.



```
USE world;
SELECT
    Continent, Region, IndepYear
```



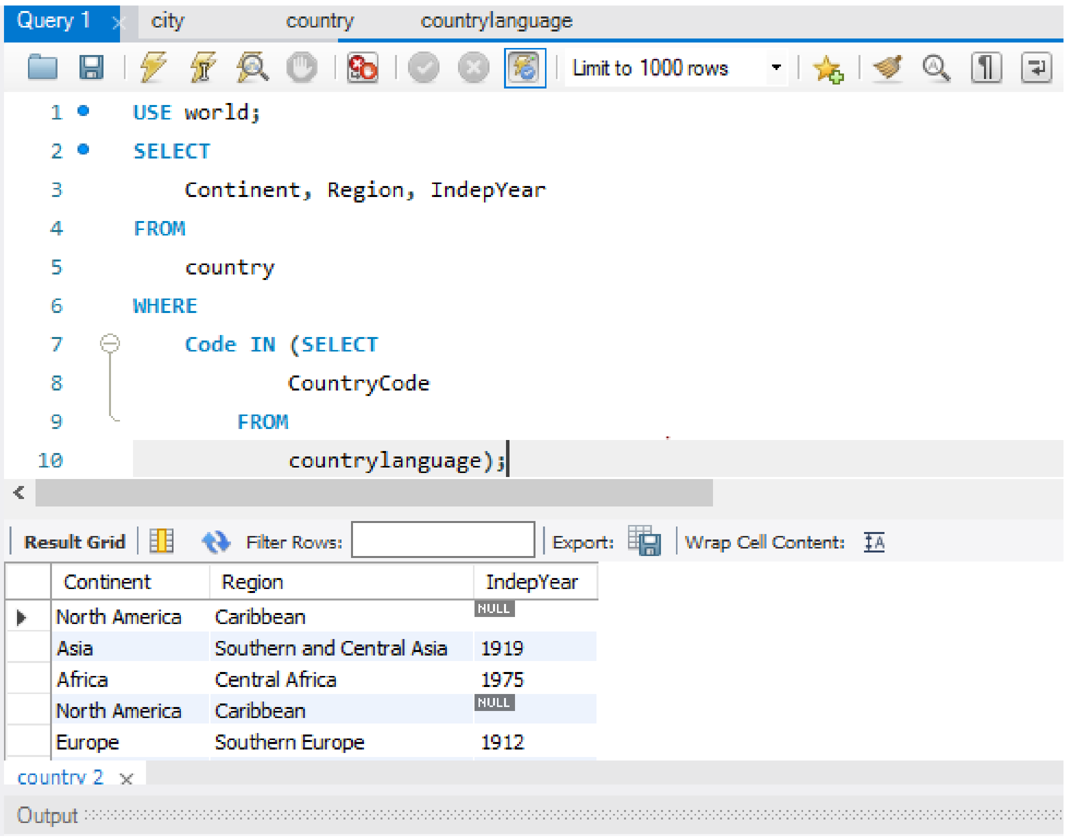
```sql
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

country 2 ×

Output



Query 1 × | city | country | countrylanguage

Limit to 1000 rows

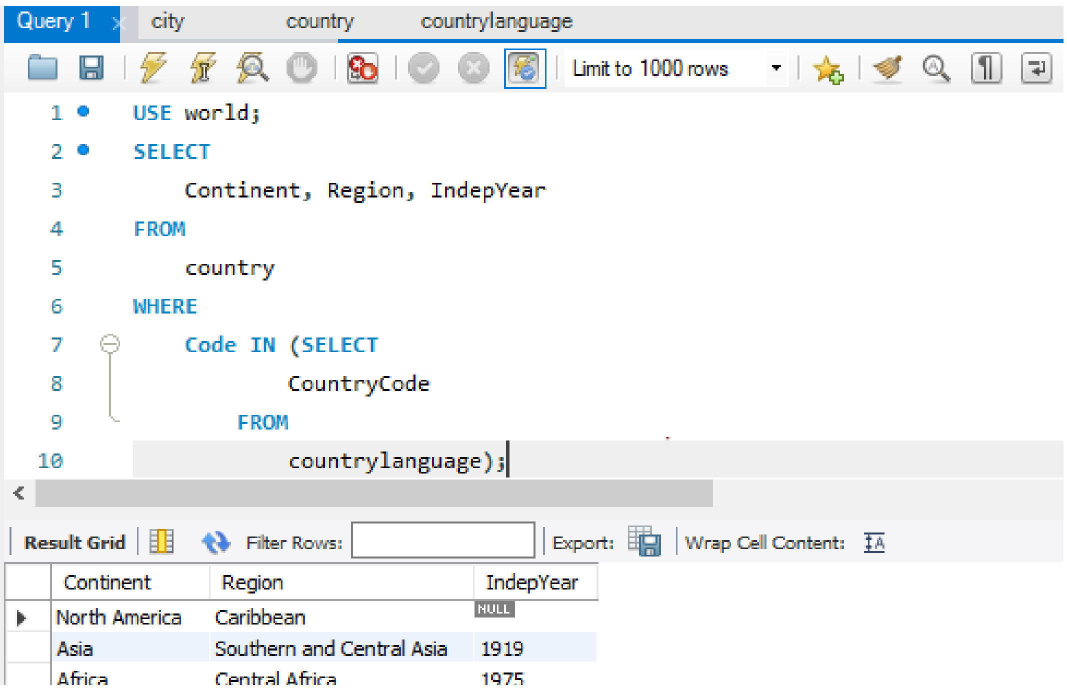
```sql
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

country 2 ×

Output



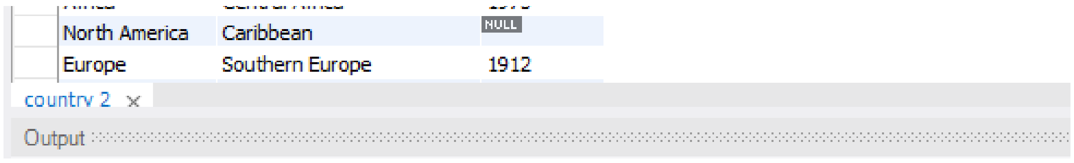
Query 1 × | city | country | countrylanguage

Limit to 1000 rows

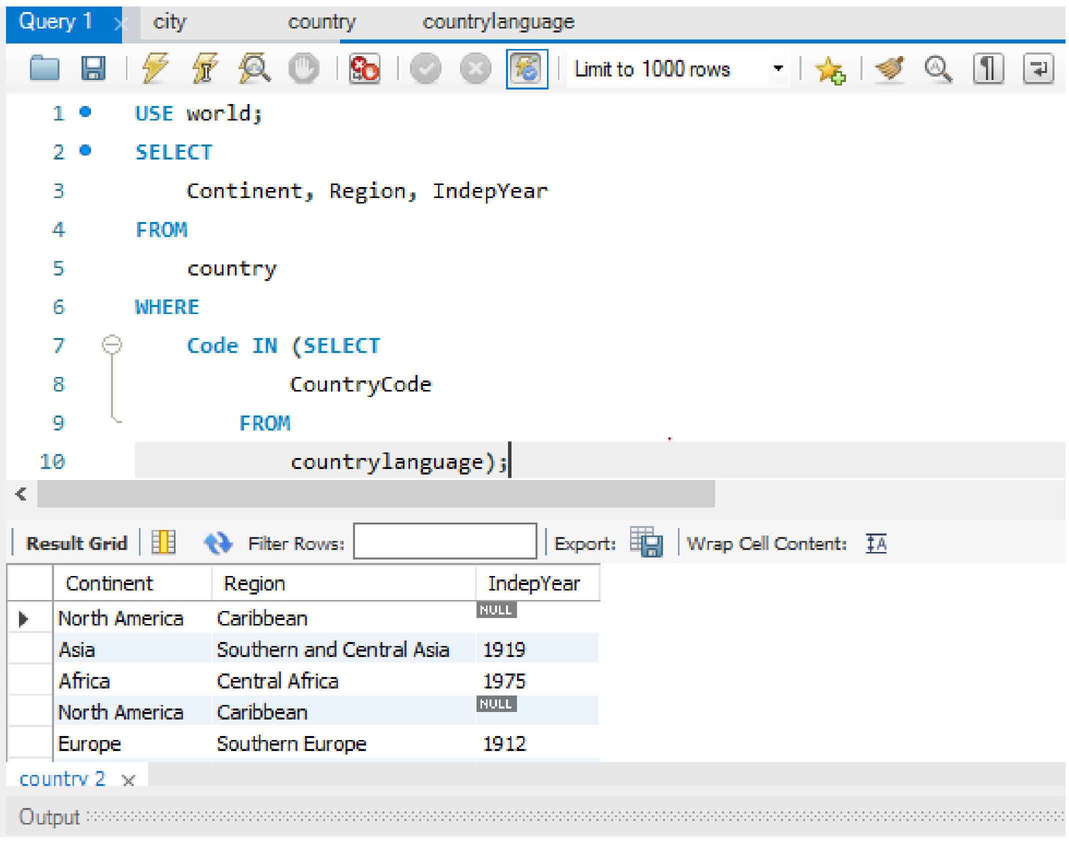
```sql
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975



North America	Caribbean	NULL
Europe	Southern Europe	1912

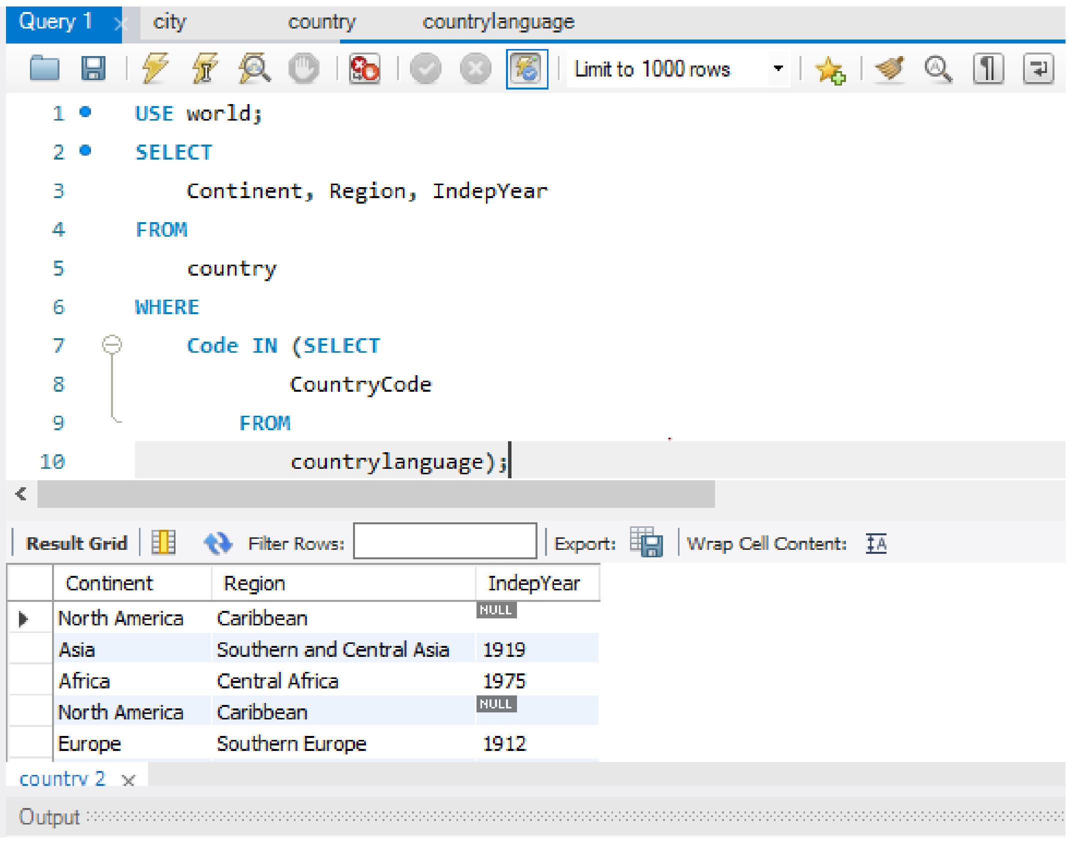


```
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

3. Correlated Sub-queries

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



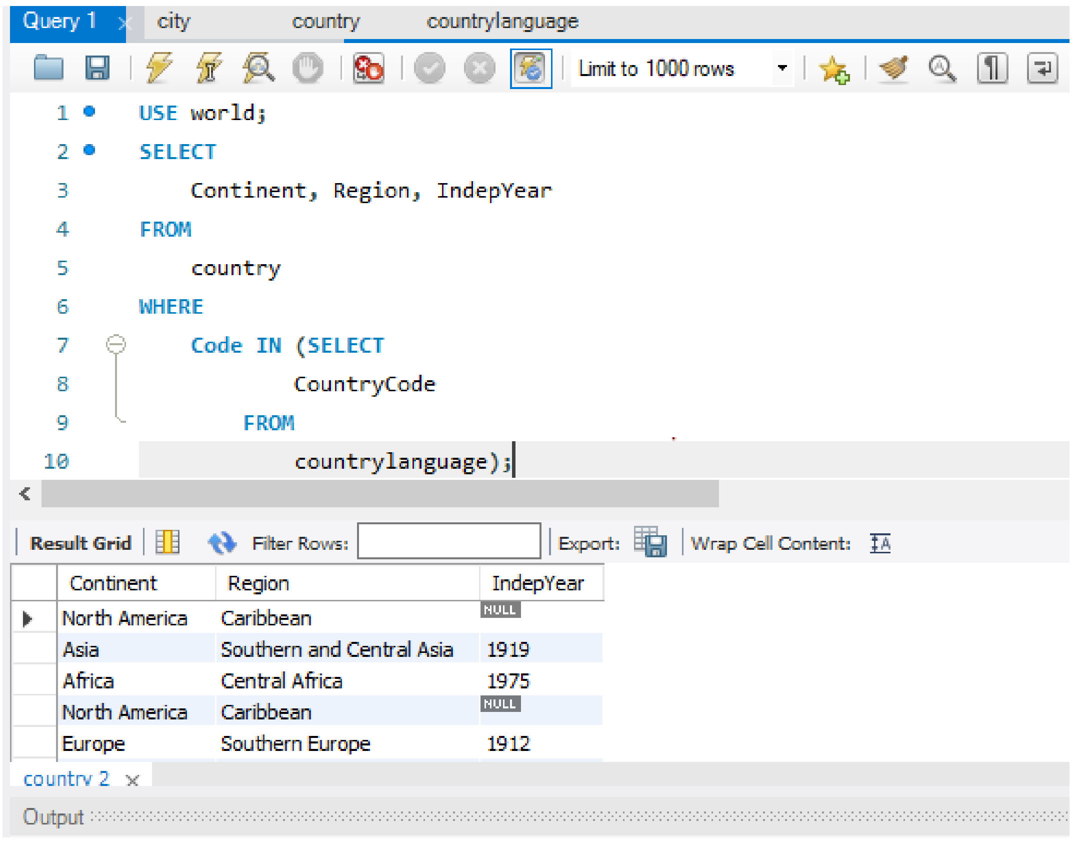
```
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement

can be a SELECT, UPDATE, or DELETE statement.

A correlated subquery is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result or set of results for each candidate row considered by the main query. In other words, you can use a correlated subquery to answer a multipart question whose answer depends on the value in each row processed by the parent statement.



```
USE world;
SELECT
    Continent, Region, IndepYear
FROM
    country
WHERE
    Code IN (SELECT
            CountryCode
        FROM
            countrylanguage);
```

Continent	Region	IndepYear
North America	Caribbean	NULL
Asia	Southern and Central Asia	1919
Africa	Central Africa	1975
North America	Caribbean	NULL
Europe	Southern Europe	1912

In Top To Down approach

For the each record of parent table,Inear Query will run 4 time in our case.

Like for each record of employee of EMP table the inner query will run for 4 times for matching condition of department table. .

A correlated SQL subquery is just a subquery that is executed many times—once for each record (row) returned by the outer (main) query. In other words, the outer query returns a table with multiple rows; the inner query then runs once for each of those rows. If your outer query returns 10 rows, then the inner query will run 10 times. And if your outer query returns 100 rows, the inner query will run 100 times.

https://learnsql.com/blog/correlated-sql-subquery-5-minutes/

4.Nested Sub-Queries

A subquery can be nested inside other subqueries. SQL has an ability to nest queries within one another. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results. SQL executes innermost subquery first, then next level.

https://www.w3resource.com/sql/subqueries/nested-subqueries.php

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. ... A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.

In [ ]:

Link :- https://www.mysqltutorial.org/mysql-triggers.aspx

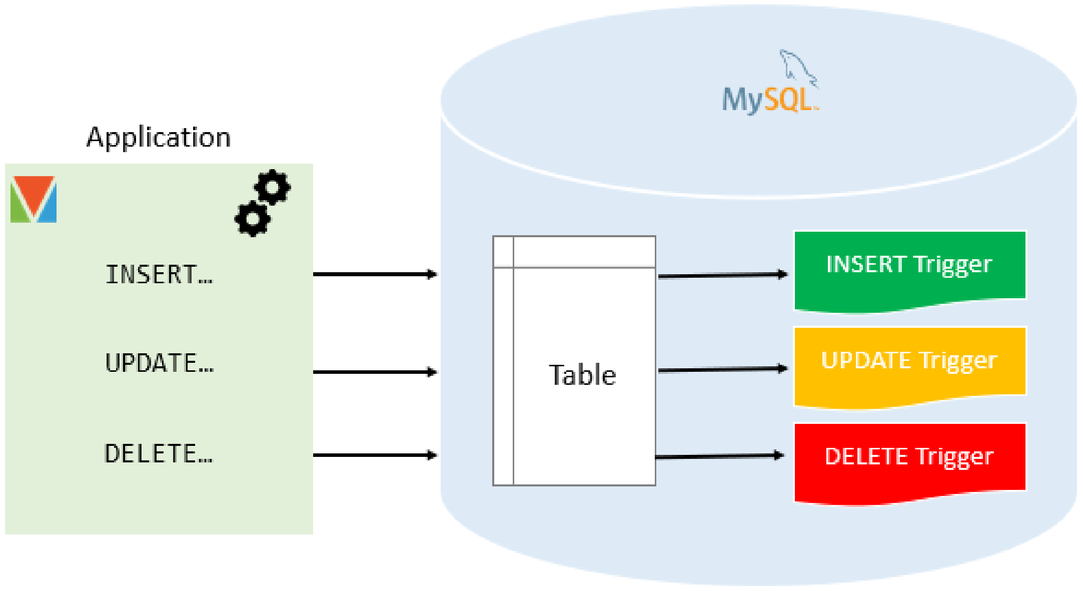
In MySQL, a trigger is a stored program invoked automatically in response to an event such as insert, update, or delete that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

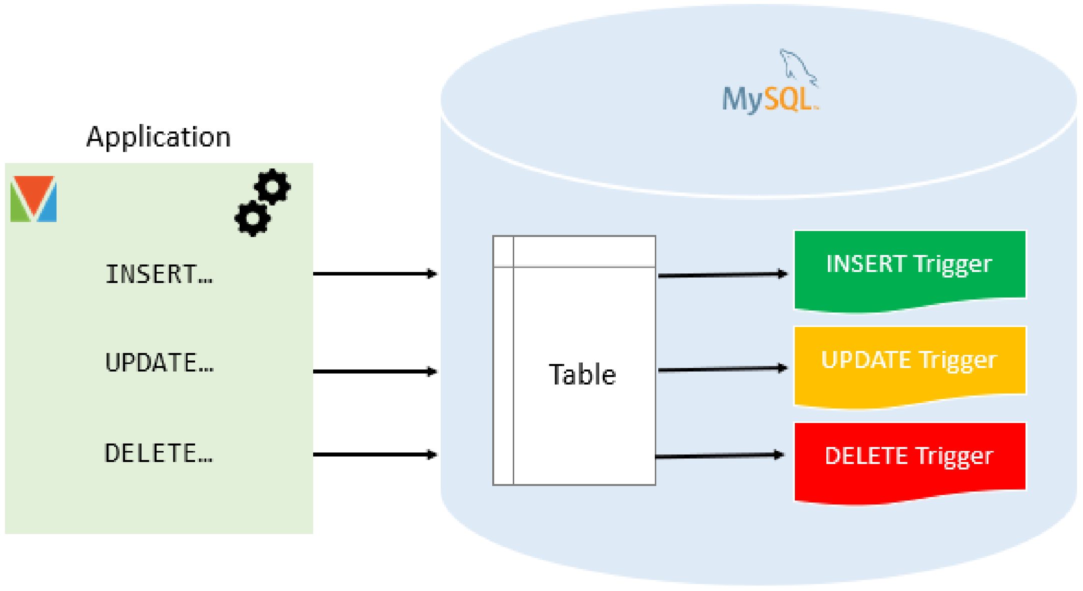
MySQL supports triggers that are invoked in response to the `INSERT, UPDATE or DELETE` **event.**

The SQL standard defines two types of triggers: row-level triggers and statement-level triggers.

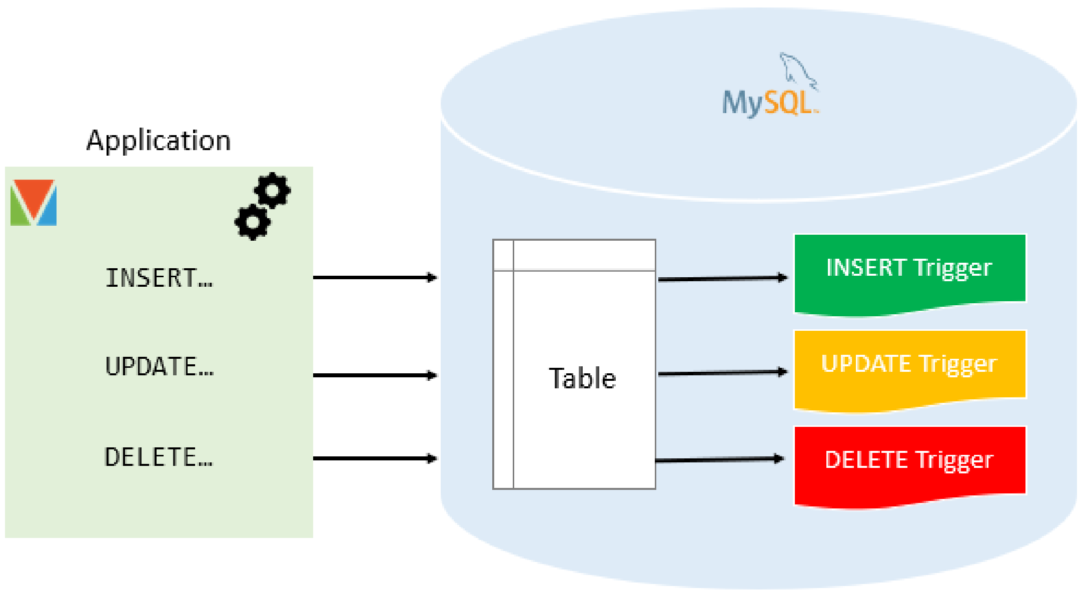
- **A row-level trigger is activated for each row that is inserted, updated, or deleted. For example, if a table has 100 rows inserted, updated, or deleted, the trigger is automatically invoked 100 times for the 100 rows affected.**
- **A statement-level trigger is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.**

MySQL supports only row-level triggers. It doesn't support statement-level triggers.

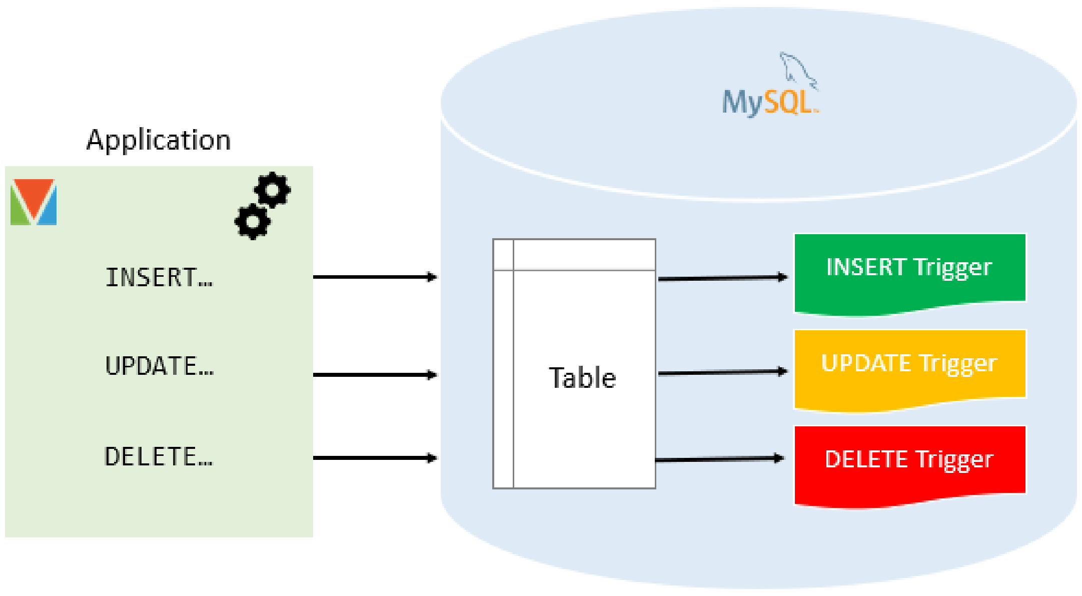


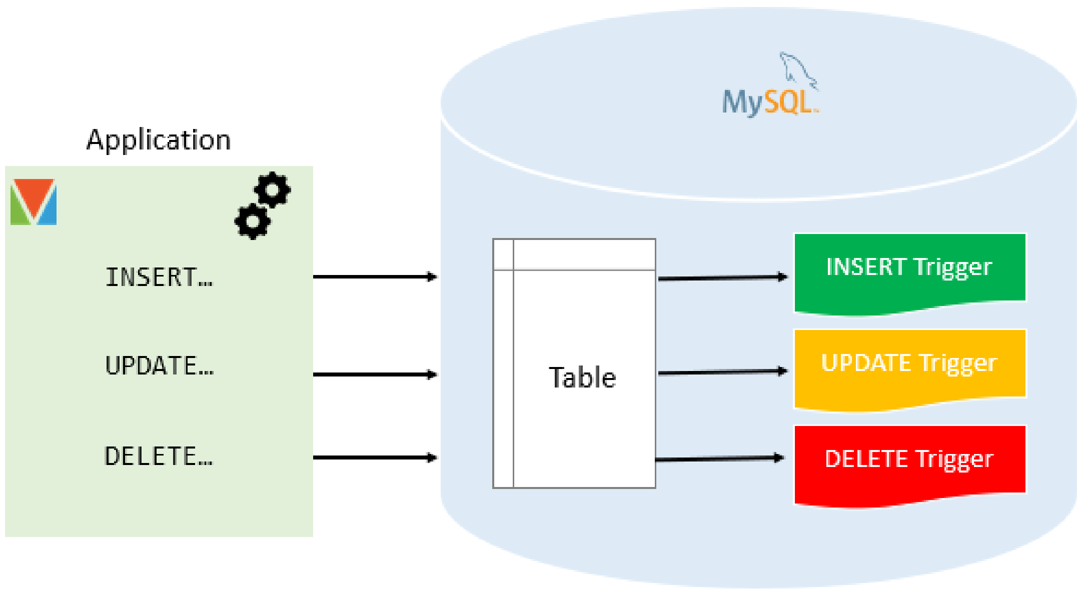


Manageing MySQL Trigger



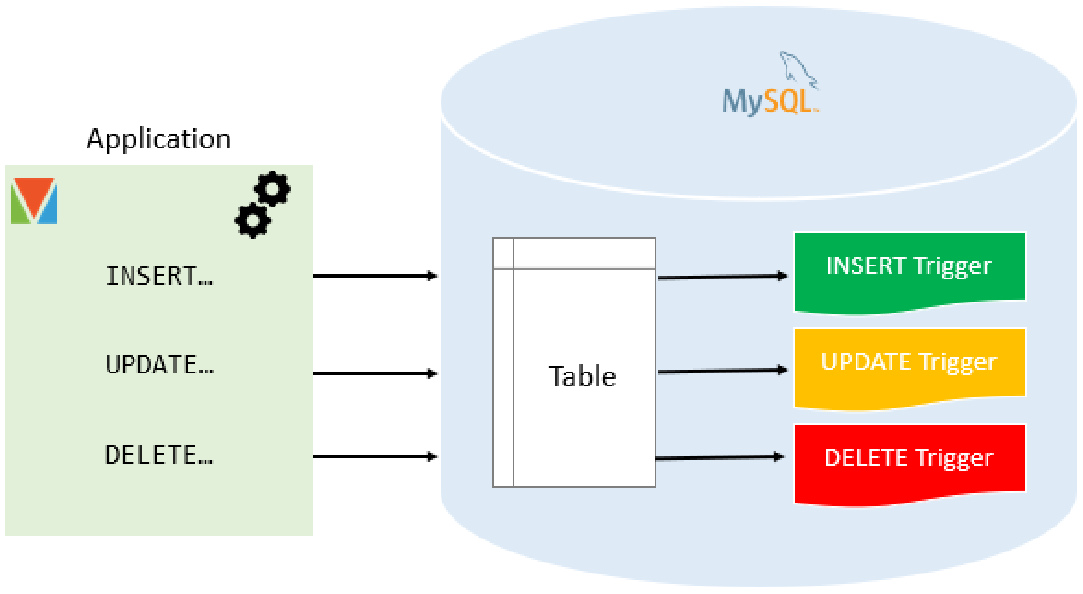
Additional Information





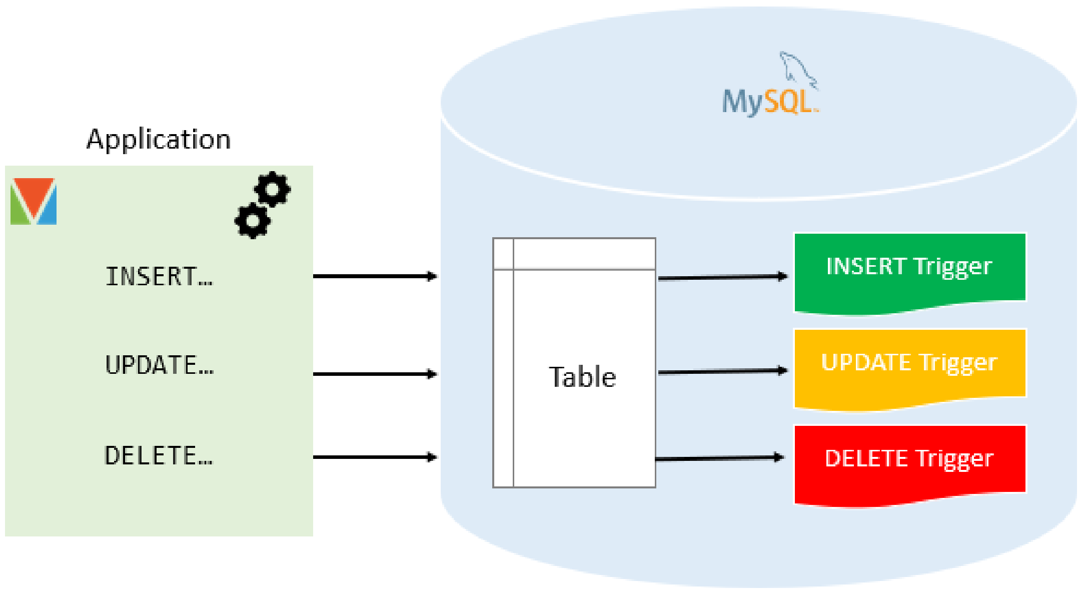
Example No.1

BEFORE INSERT



Example 2

BEFORE UPDATE



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

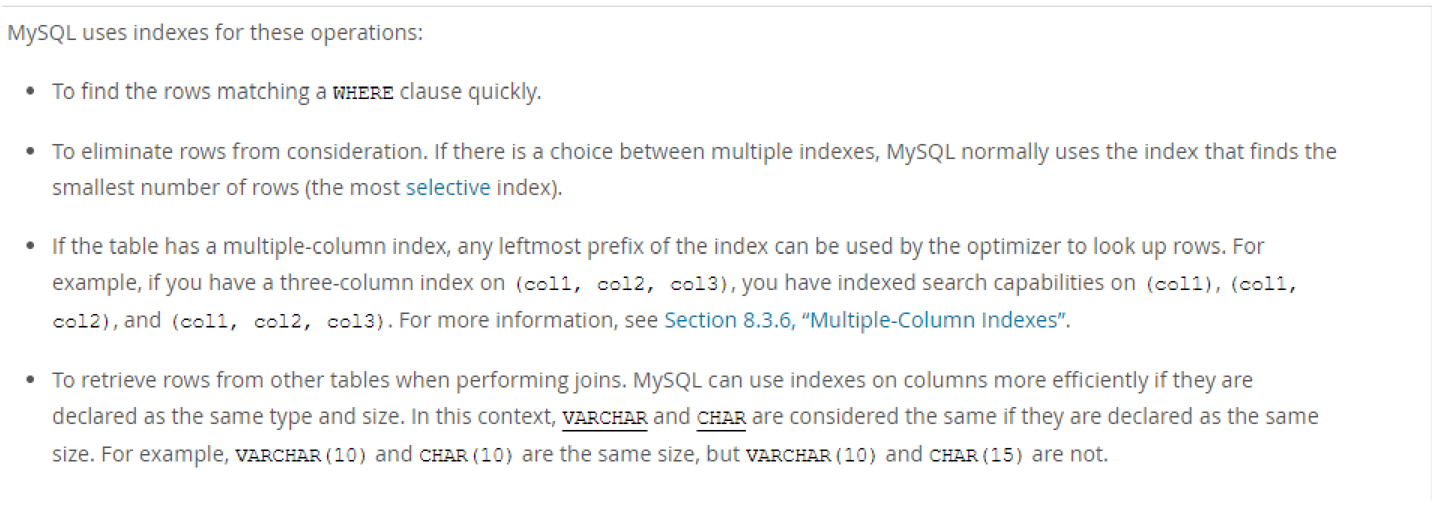
```
```

In [ ]:

```
```

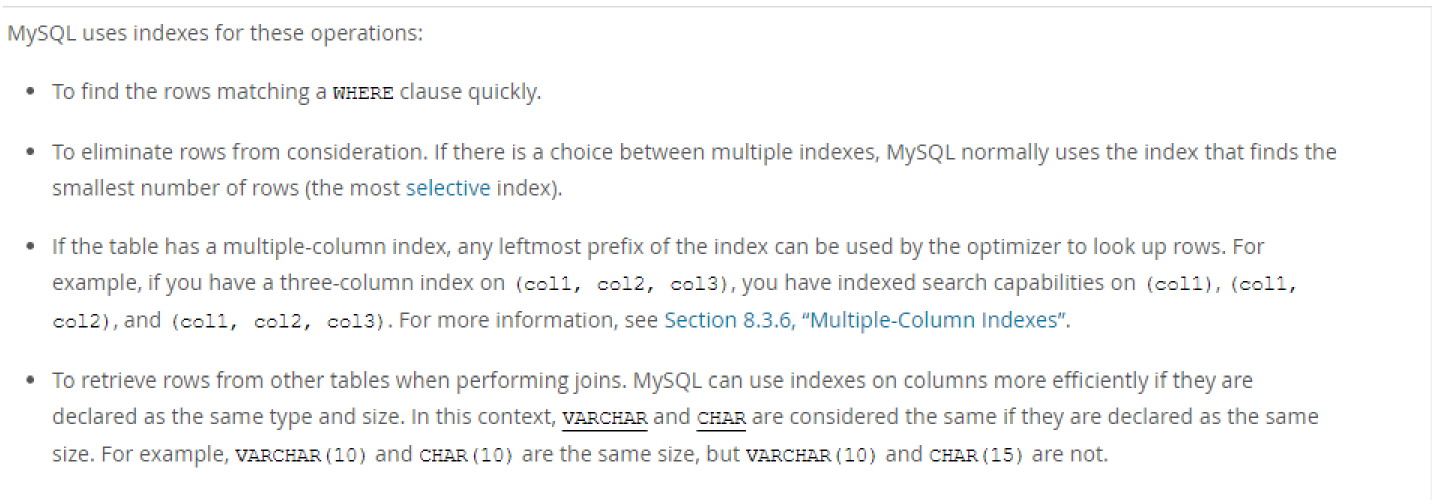
Introduction About Indexes.

- **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 this 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.**

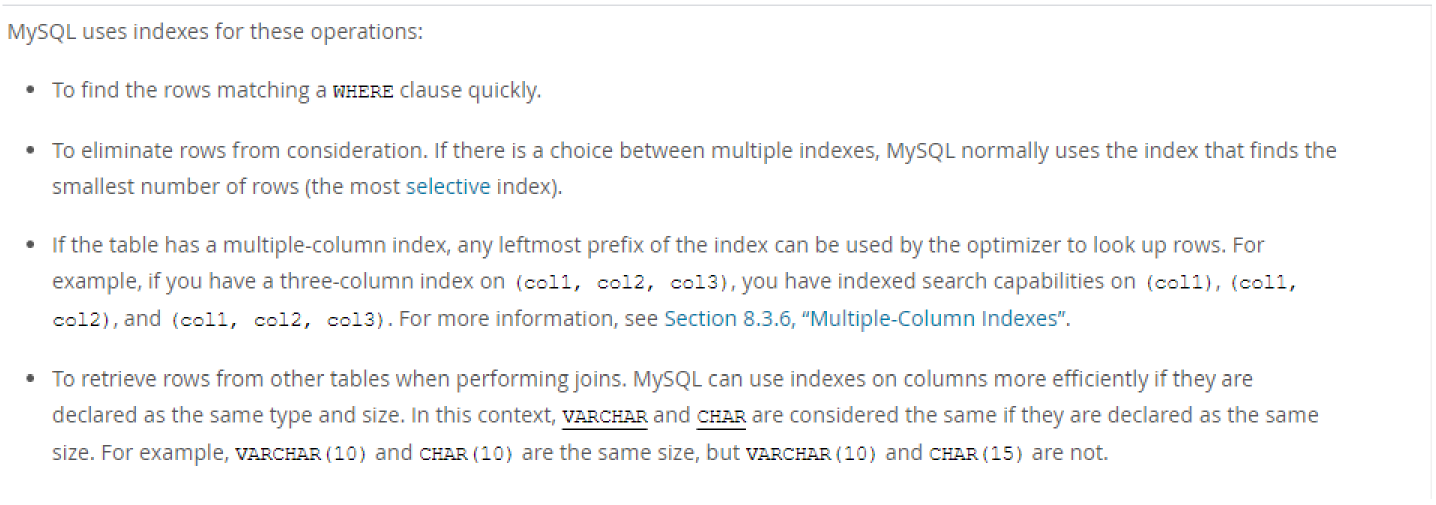

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, “Multiple-Column Indexes”.
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

Some Important Things About MySQL.

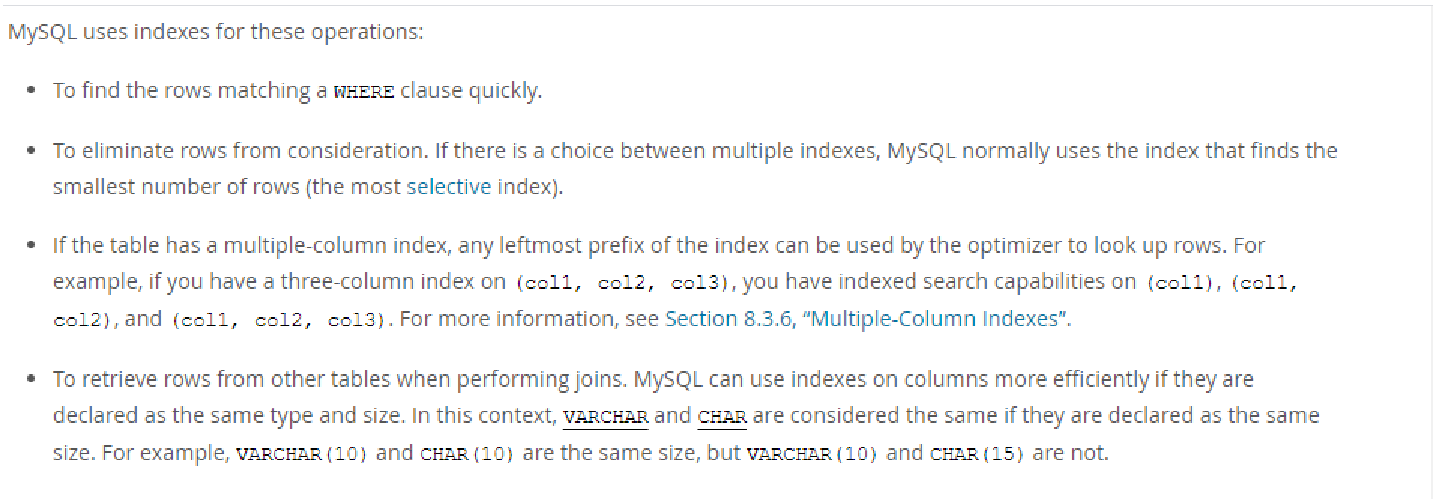

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, “Multiple-Column Indexes”.
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.


MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, “Multiple-Column Indexes”.
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

Example :-



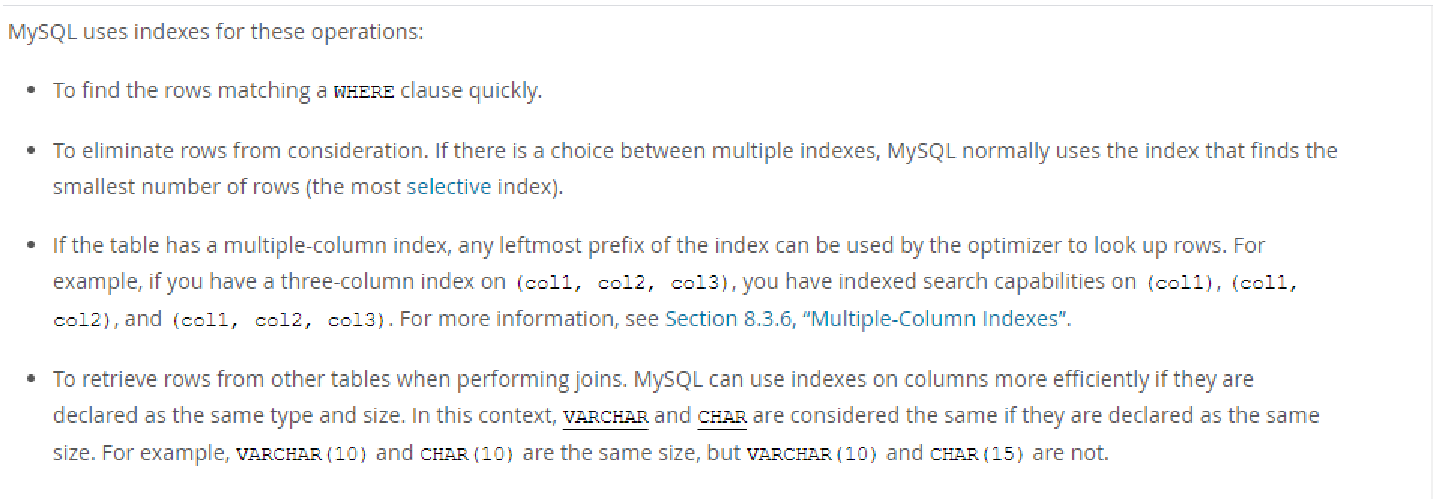
MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, "Multiple-Column Indexes".
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

The above example while running the code,it takes the 0.078 sec.

But if we use the INDEX and then the above then see how time it take for an execution.

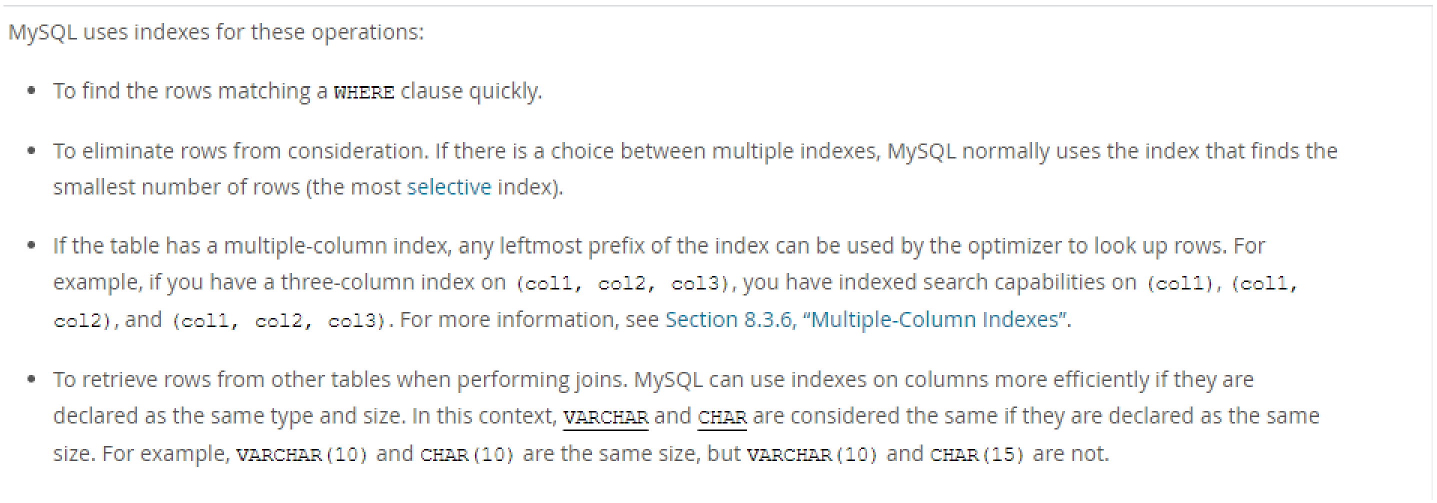
1.Single Column Indexes.



MySQL uses indexes for these operations:

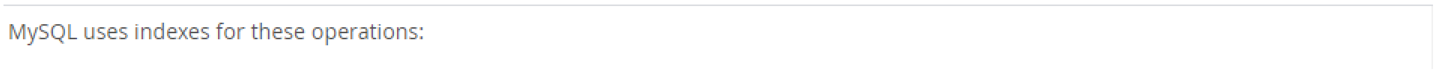
- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, "Multiple-Column Indexes".
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

2. Composite Indexes.



MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, "Multiple-Column Indexes".
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.



MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, "Multiple-Column Indexes".
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, "Multiple-Column Indexes".
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, "Multiple-Column Indexes".
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

How to DROP INDEX of Table.

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, "Multiple-Column Indexes".
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

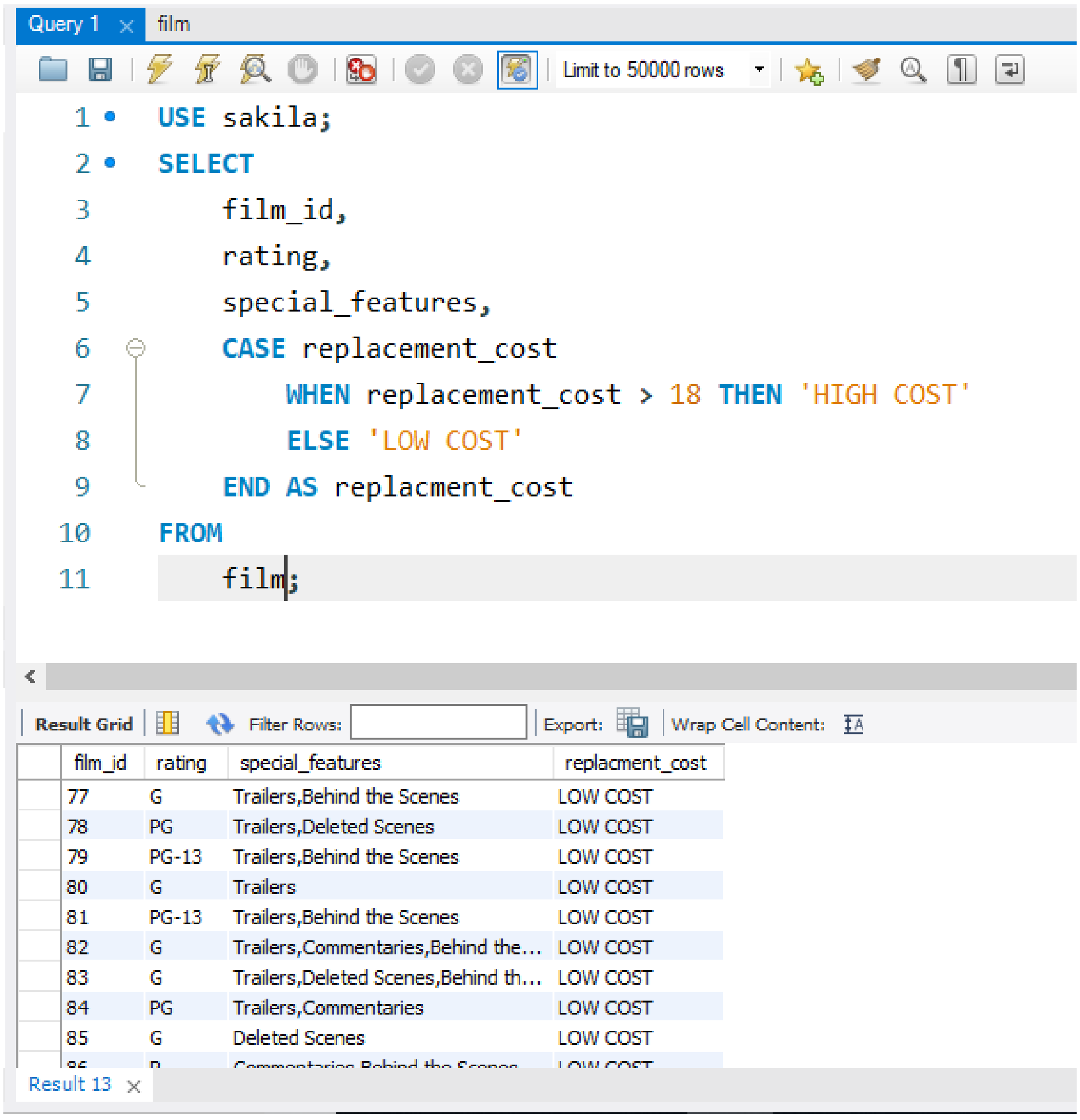
```
In [ ]:
```

Thank You !!

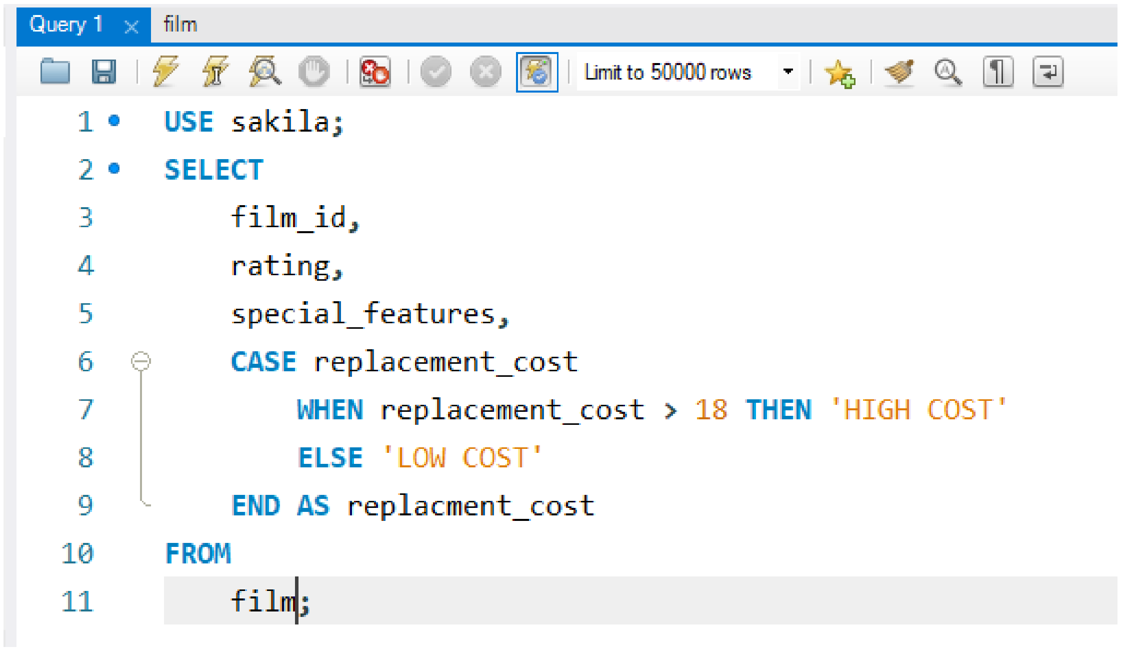
CASE Statement

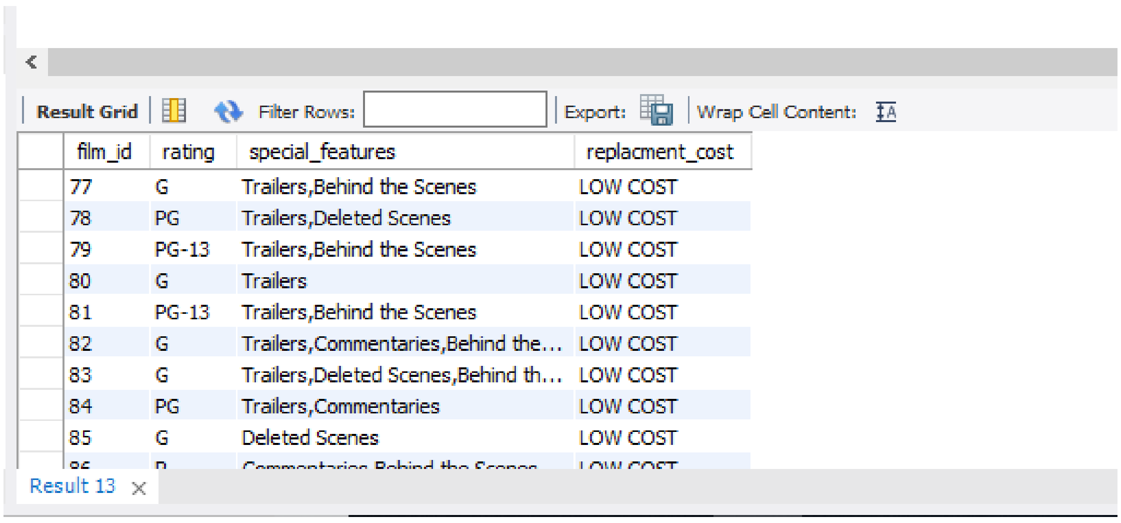
https://www.mysqltutorial.org/mysql-case-function/

https://www.w3schools.com/mysql/func_mysql_case.asp


```
USE sakila;
SELECT
    film_id,
    rating,
    special_features,
    CASE replacement_cost
        WHEN replacement_cost > 18 THEN 'HIGH COST'
        ELSE 'LOW COST'
    END AS replacment_cost
FROM
    film;
```

film_id	rating	special_features	replacment_cost
77	G	Trailers,Behind the Scenes	LOW COST
78	PG	Trailers,Deleted Scenes	LOW COST
79	PG-13	Trailers,Behind the Scenes	LOW COST
80	G	Trailers	LOW COST
81	PG-13	Trailers,Behind the Scenes	LOW COST
82	G	Trailers,Commentaries,Behind the...	LOW COST
83	G	Trailers,Deleted Scenes,Behind th...	LOW COST
84	PG	Trailers,Commentaries	LOW COST
85	G	Deleted Scenes	LOW COST

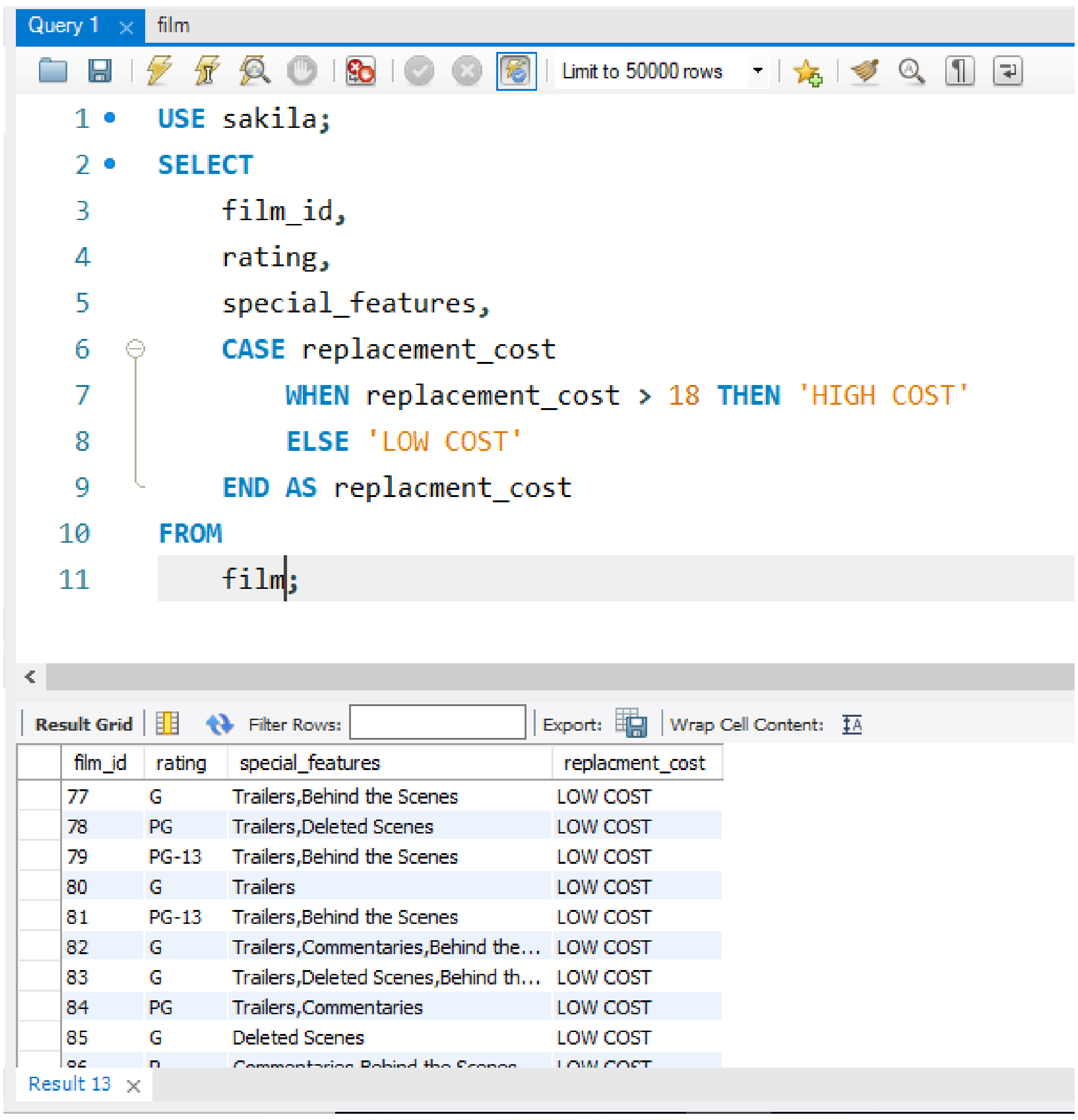

```
USE sakila;
SELECT
    film_id,
    rating,
    special_features,
    CASE replacement_cost
        WHEN replacement_cost > 18 THEN 'HIGH COST'
        ELSE 'LOW COST'
    END AS replacment_cost
FROM
    film;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

film_id	rating	special_features	replacment_cost
77	G	Trailers,Behind the Scenes	LOW COST
78	PG	Trailers,Deleted Scenes	LOW COST
79	PG-13	Trailers,Behind the Scenes	LOW COST
80	G	Trailers	LOW COST
81	PG-13	Trailers,Behind the Scenes	LOW COST
82	G	Trailers,Commentaries,Behind the...	LOW COST
83	G	Trailers,Deleted Scenes,Behind th...	LOW COST
84	PG	Trailers,Commentaries	LOW COST
85	G	Deleted Scenes	LOW COST

Result 13



Query 1 | film

Limit to 50000 rows

```sql
USE sakila;
SELECT
    film_id,
    rating,
    special_features,
    CASE replacement_cost
        WHEN replacement_cost > 18 THEN 'HIGH COST'
        ELSE 'LOW COST'
    END AS replacment_cost
FROM
    film;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

film_id	rating	special_features	replacment_cost
77	G	Trailers,Behind the Scenes	LOW COST
78	PG	Trailers,Deleted Scenes	LOW COST
79	PG-13	Trailers,Behind the Scenes	LOW COST
80	G	Trailers	LOW COST
81	PG-13	Trailers,Behind the Scenes	LOW COST
82	G	Trailers,Commentaries,Behind the...	LOW COST
83	G	Trailers,Deleted Scenes,Behind th...	LOW COST
84	PG	Trailers,Commentaries	LOW COST
85	G	Deleted Scenes	LOW COST

Result 13

Thanks

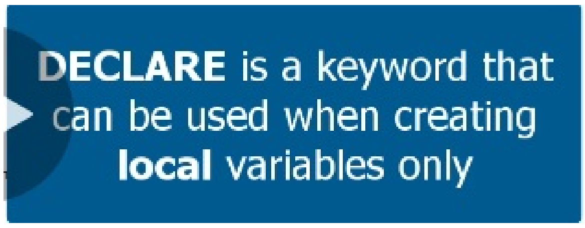
Types Of Variable In SQL

There are three types of variables

- **Local Variable.**
- **Session Variable.**
- **Gloabal Variable.**

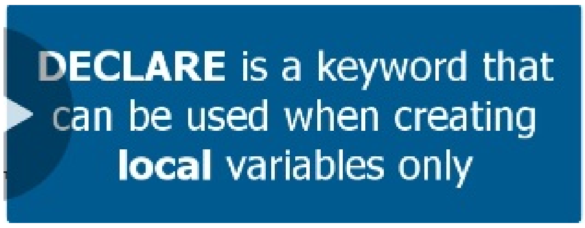
1. Local Variable.

- **The variable that is visible only in the `BEGIN-END` block in which it is created.**
- **By default, a local variable starts with `@`.**
- **Every local variable scope has the restriction to the current batch or procedure within any given session.**


DECLARE is a keyword that can be used when creating **local** variables only

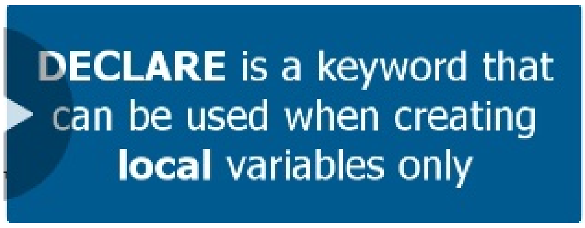
Any local variable cannot accessed from outside the `BEGIN-END` block.

Example :-


DECLARE is a keyword that can be used when creating **local** variables only

Here we have variable which is nothing but the `v_avg_rc` and it is avalible inside the `BEGIN-END` block but it can not be access ouside the function if we tried we will get an error.

Look at the below we are getting an error once we have call function outside the `BEGIN-END` Block.

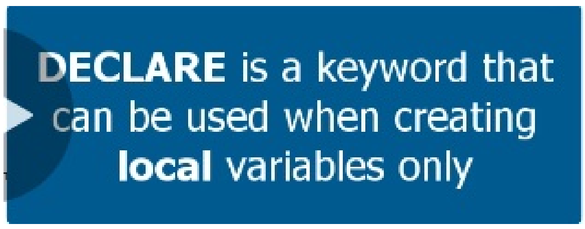

DECLARE is a keyword that can be used when creating **local** variables only

https://www.guru99.com/sql-server-variable.html

2. Session Variable.

- **The session is nothing but the series of information exchange intractions,or dialogue,between a computer and user.**

- The session begins at a certain point in time and terminates at another,later point.


DECLARE is a keyword that can be used when creating local variables only

- This variable avalible only for session in which you are opearting.
- It is defined on our server,and it lives there
- It is visible to connection being used only.
- If the 100 user connected with at a time then there will be a 100 connections and 100 sessions.

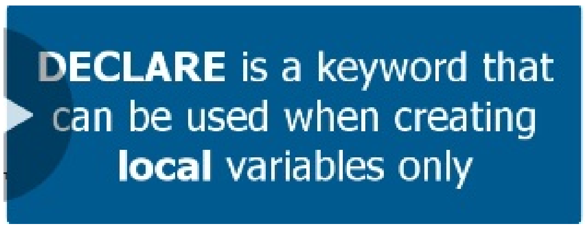
- A session variable is a user-defined variable (not a server option) that starts with @, does not require declaration, can be used in any SQL query or statement, not visible to other sessions, and exists until the end of the current session.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser. So; Session variables hold information about one single user, and are available to all pages in one application.
- A session is just a result of a successful connection . Any MySQL client requires some connection settings to establish a connection and after the connection has been established it acquires a connection id (thread id) and some context which is called session.

3. Global Variable

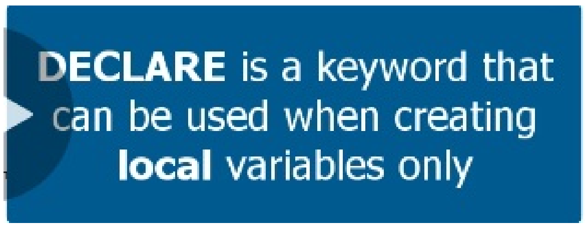
Global variables are pre-defined system functions.The server maintains the values in these variables. Global variables return various pieces of information about the current user environment for SQL Server. Global Variable are automatically updated and interact with the system

Global Varible apply to all connections related to a specific server.

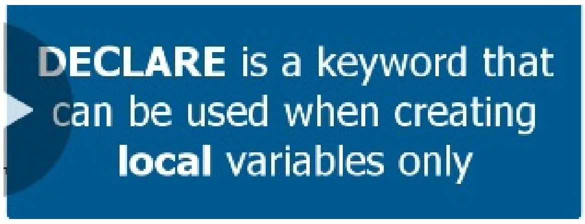
Syntax


DECLARE is a keyword that can be used when creating local variables only

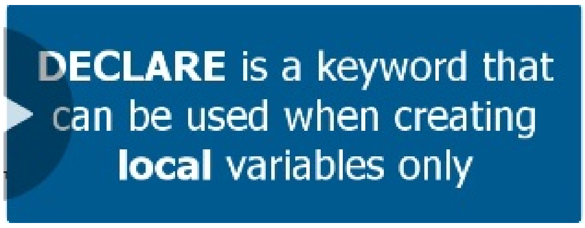
- You can not set just any variable as Global.
- A specific group of predefined variables in MySQL is suitable for this job.They are called as `system variable`.

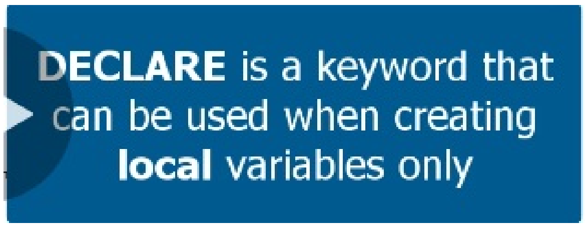

DECLARE is a keyword that can be used when creating local variables only

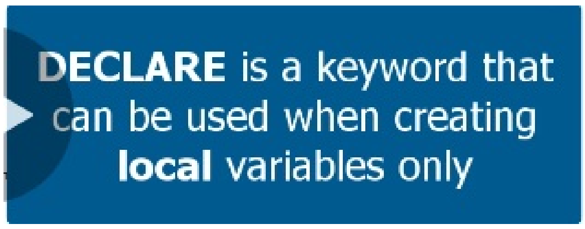
Example :-


DECLARE is a keyword that can be used when creating local variables only

USER defined Vs System variable

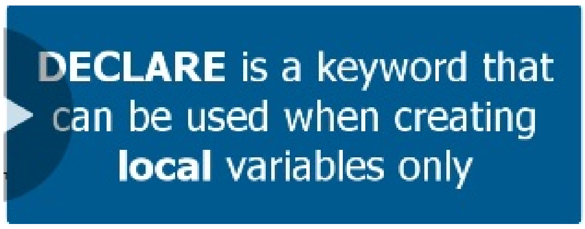

DECLARE is a keyword that can be used when creating local variables only


DECLARE is a keyword that can be used when creating local variables only


DECLARE is a keyword that can be used when creating local variables only

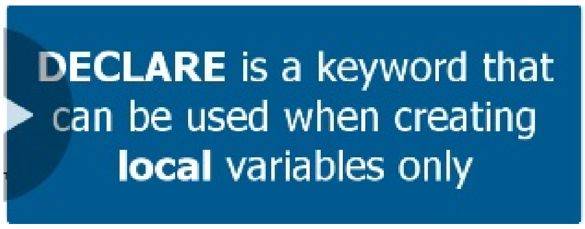
If we use the `Max_connections` Global and then use as Session after considering as the session variable we will get an error.

Because Max_connections is global variable but if we use as session variable that why it is throwing an error.


DECLARE is a keyword that can be used when creating local variables only

Lets try another GLOBAL variable that is `.sql_mode()` that is helping us to adjust the setting.

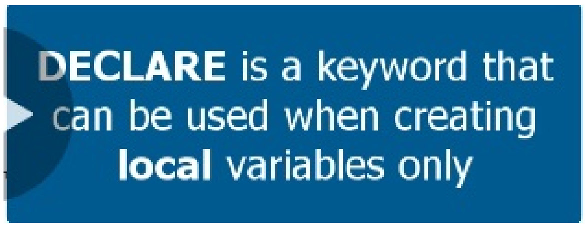
It is working for both session and Global variable,because it is not use as session variable or not going to use in any server setting.


DECLARE is a keyword that can be used when creating local variables only

Notes.

- **USER defined the LOCAL variable = Session Variable**

- USER can defined the LOCAL variable or Session Variable.
- System variable can be set as session variables or as Global Variables.
- Not all the system variable can be set as session.


DECLARE is a keyword that can be used when creating **local** variables only

Thank You !!

About Variable In SQL

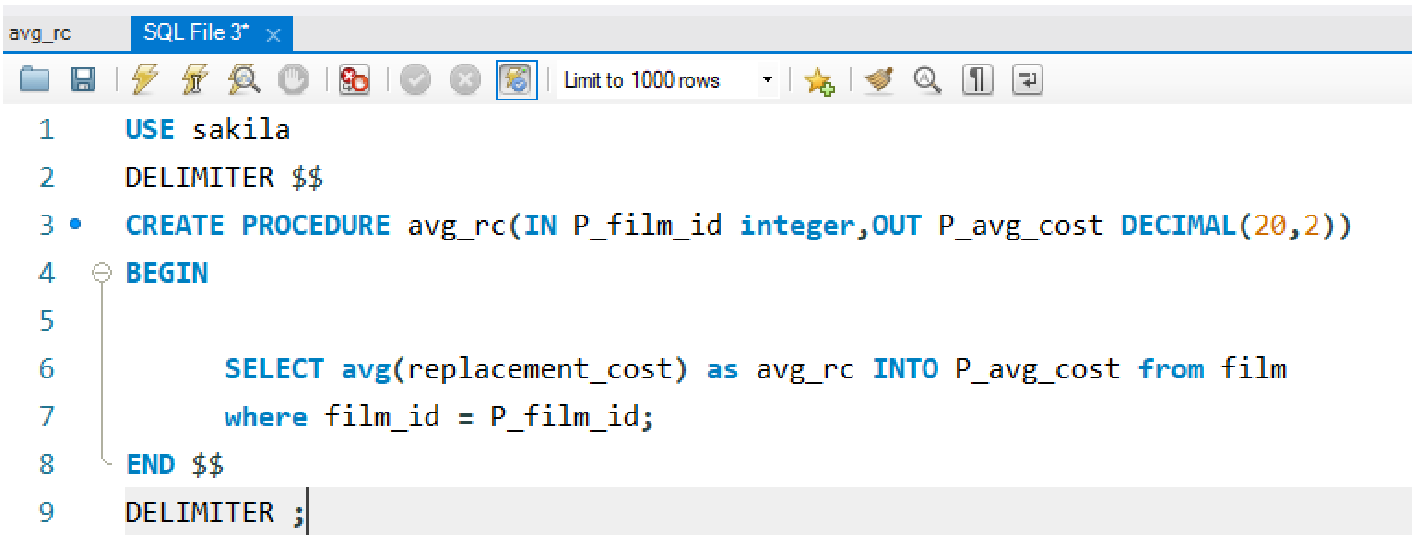
- SQL Server provides us with two methods in T-SQL to assign a value to a previously created local SQL variable.
- The first method is the SET statement, the ANSI standard statement that is commonly used for variable value assignment.
- The second statement is the SELECT statement. In addition to its main usage to form the logic that is used to retrieve data from a database table or multiple tables in SQL Server, the SELECT statement can be used also to assign a value to a previously created local variable directly or from a variable, view or table.
- Although both T-SQL statements fulfill the SQL variable value assignment task, there is a number of differences between the SET and SELECT statements that may lead you to choose one of them in specific circumstances, over the other.

- Variables in SQL procedures are defined by using the DECLARE statement. Values can be assigned to variables using the SET statement or the SELECT INTO statement or as a default value when the variable is declared. Literals, expressions, the result of a query, and special register values can be assigned to variables.

Example On Variable

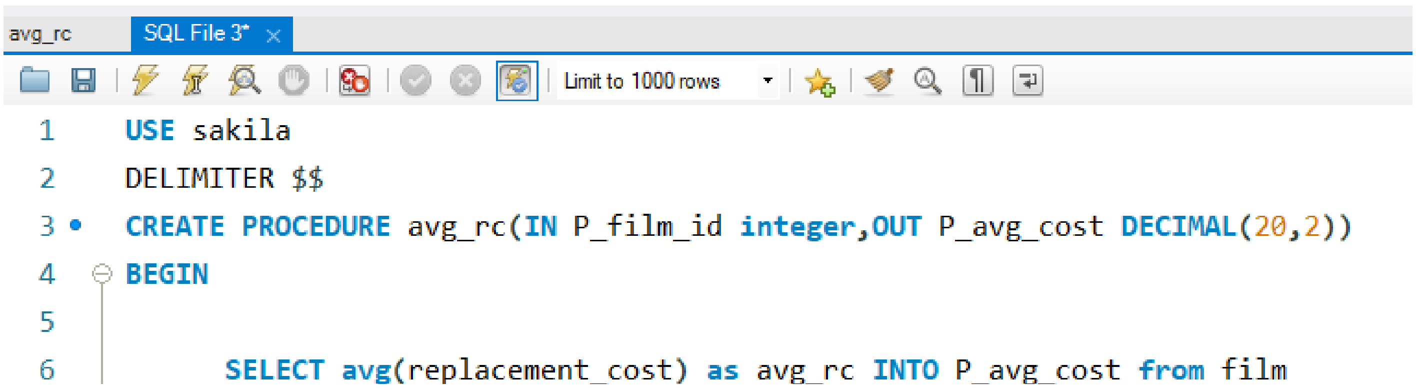
Example 1

We have created the one stored procedure and then ran and we will get an output as below way



```
USE sakila
DELIMITER $$
CREATE PROCEDURE avg_rc(IN P_film_id integer,OUT P_avg_cost DECIMAL(20,2))
BEGIN

	SELECT avg(replacement_cost) as avg_rc INTO P_avg_cost from film
	where film_id = P_film_id;
END $$
DELIMITER ;
```

Result



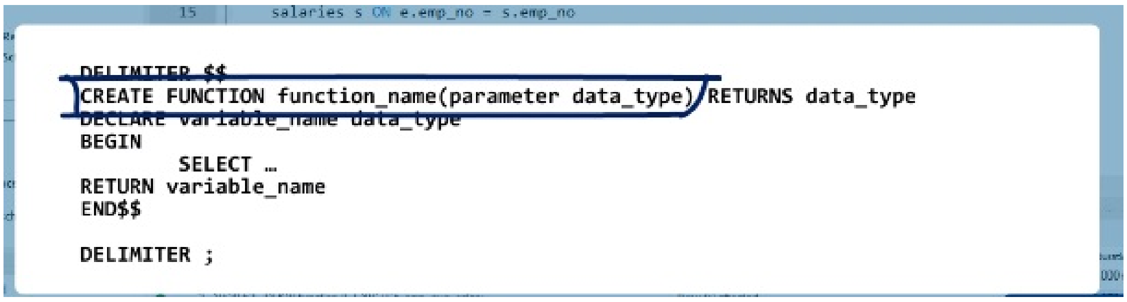
```
USE sakila
DELIMITER $$
CREATE PROCEDURE avg_rc(IN P_film_id integer,OUT P_avg_cost DECIMAL(20,2))
BEGIN

	SELECT avg(replacement_cost) as avg_rc INTO P_avg_cost from film
```

```
        where film_id = P_film_id;
END $$
DELIMITER ;
```

- In this case if we can see clearly we have first use `SET` keyword to declear the variable and it is always prefer to use `@` at begining of any variable which we want to declear.
- After that we have use the variable inside the stored procedure with input parameter is `film_id` and output as variable that is `P_avg_cost`.
- Finally,For retrieving the data we are select statement with variable name.

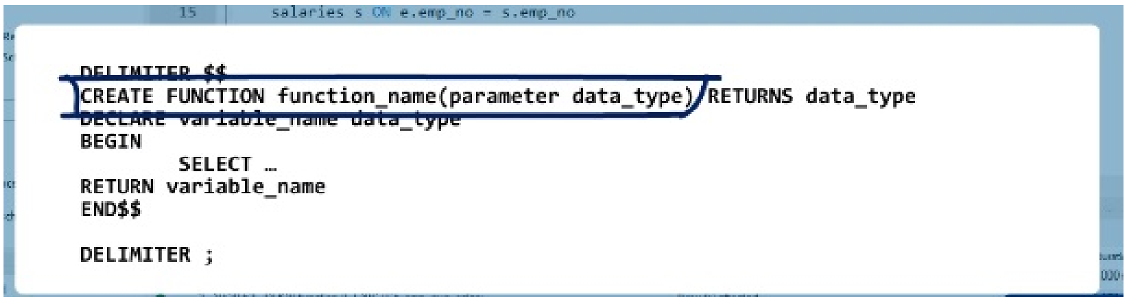
User Defined Functions In MySQL

The syntax we are using inside the function which is given below way that is showing how to use query in order built the function.



```
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
        SELECT …
RETURN variable_name
END$$

DELIMITER ;
```

Result code Function.

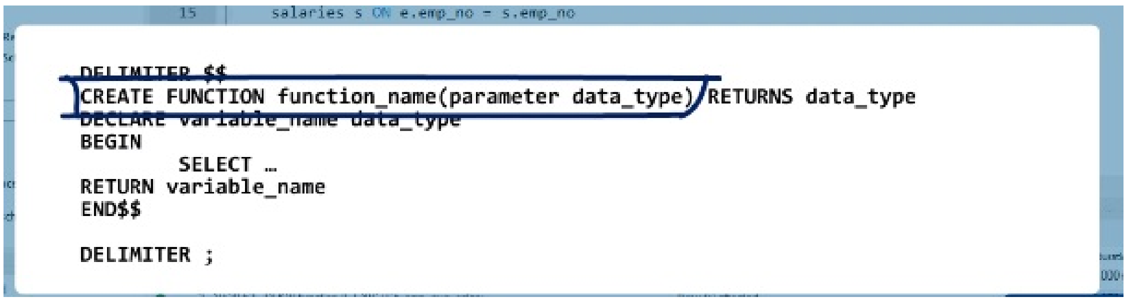


```
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
        SELECT …
RETURN variable_name
END$$

DELIMITER ;
```

In order to get rid from the `ERROR` we have to us the `DETERMINISTIC` Keyword inside the function.

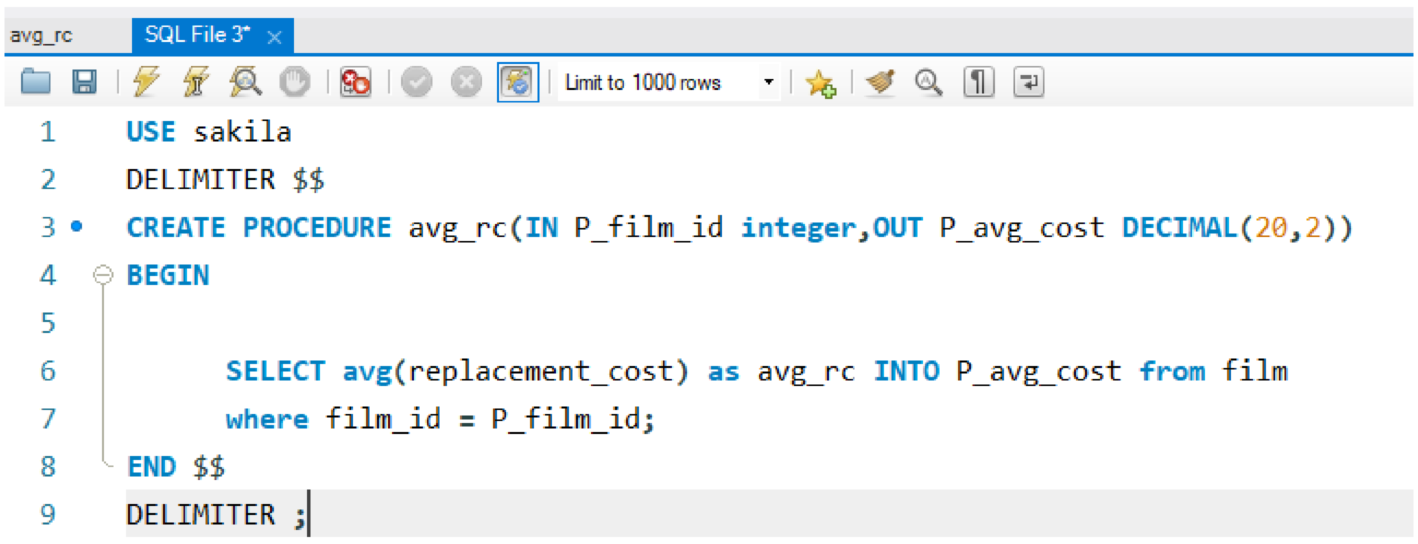
Difference Between Stored Procedures and User Defined Function.

Technical Difference

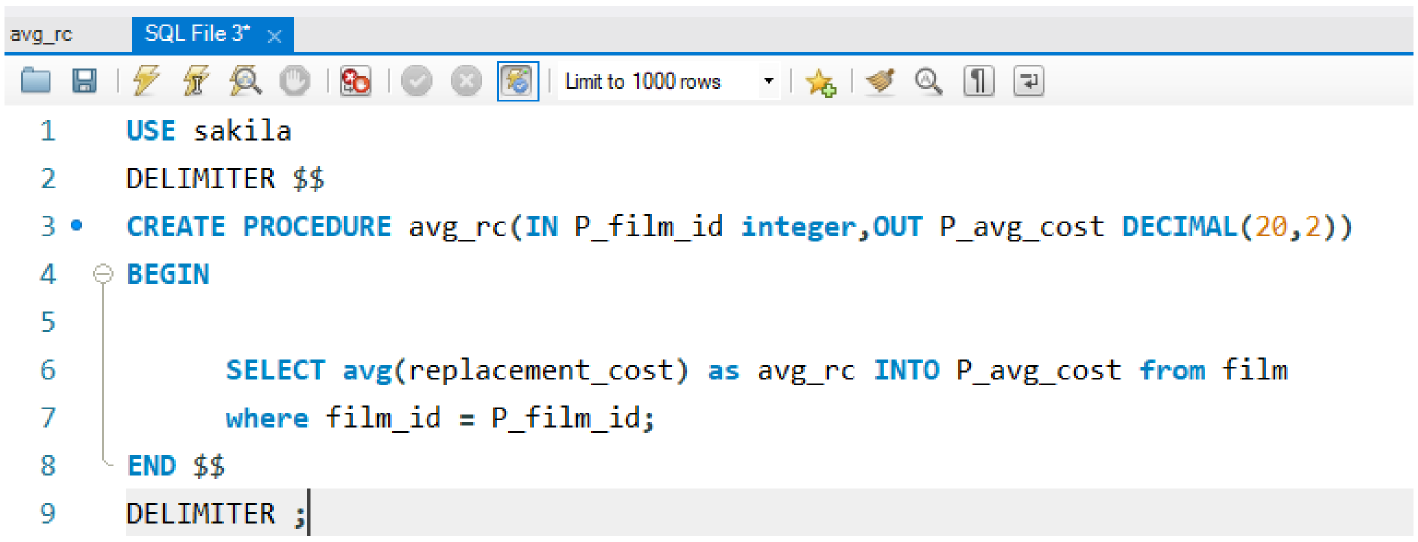


```
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
        SELECT …
RETURN variable_name
END$$

DELIMITER ;
```

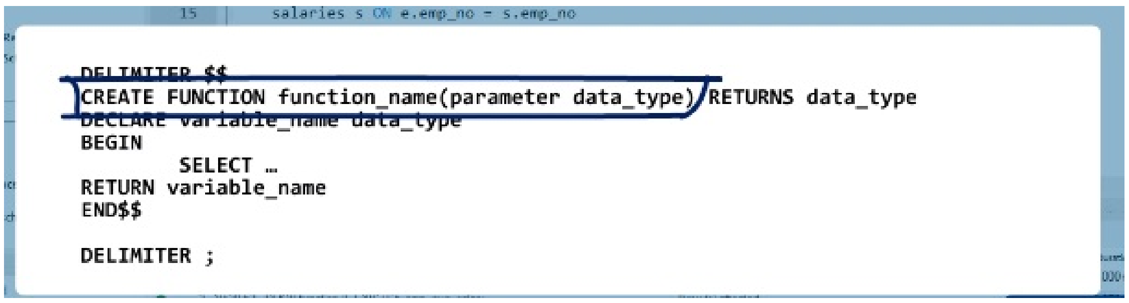
Conceptual Difference



```
USE sakila
DELIMITER $$
CREATE PROCEDURE avg_rc(IN P_film_id integer,OUT P_avg_cost DECIMAL(20,2))
BEGIN

        SELECT avg(replacement_cost) as avg_rc INTO P_avg_cost from film
        where film_id = P_film_id;
END $$
DELIMITER ;
```



```
USE sakila
DELIMITER $$
CREATE PROCEDURE avg_rc(IN P_film_id integer,OUT P_avg_cost DECIMAL(20,2))
BEGIN

        SELECT avg(replacement_cost) as avg_rc INTO P_avg_cost from film
        where film_id = P_film_id;
END $$
DELIMITER ;
```



```
DELIMITER $$
CREATE FUNCTION function_name(parameter data_type) RETURNS data_type
DECLARE variable_name data_type
BEGIN
        SELECT …
RETURN variable_name
END$$

DELIMITER ;
```

Thank You !!

Introduction About The Stored Routine.

- When we have mutiple user and they want to use same query everytime at particular session.Then it will difficult to write a code everytime and need to remember that code.
- In order to remove this kind of difficulty we have an option that is Stored Routine.
- MySQL supports stored routines (procedures and functions). A stored routine is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored routine instead.

- An SQL-invoked routine (or SQL routine), is the generic name for either a procedure (SQL-invoked procedure) or a function (SQL-invoked function). SQL routines are dependent on some Schema (they're also called Schema-level routines) and are created, altered and dropped using standard SQL statements.
- A stored routine is held on the database server, rather than in the application. For applications based on a client-server architecture, calling a stored routine is faster and requires less network bandwidth than transmitting an entire series of SQL statements and taking decisions on the result sets.

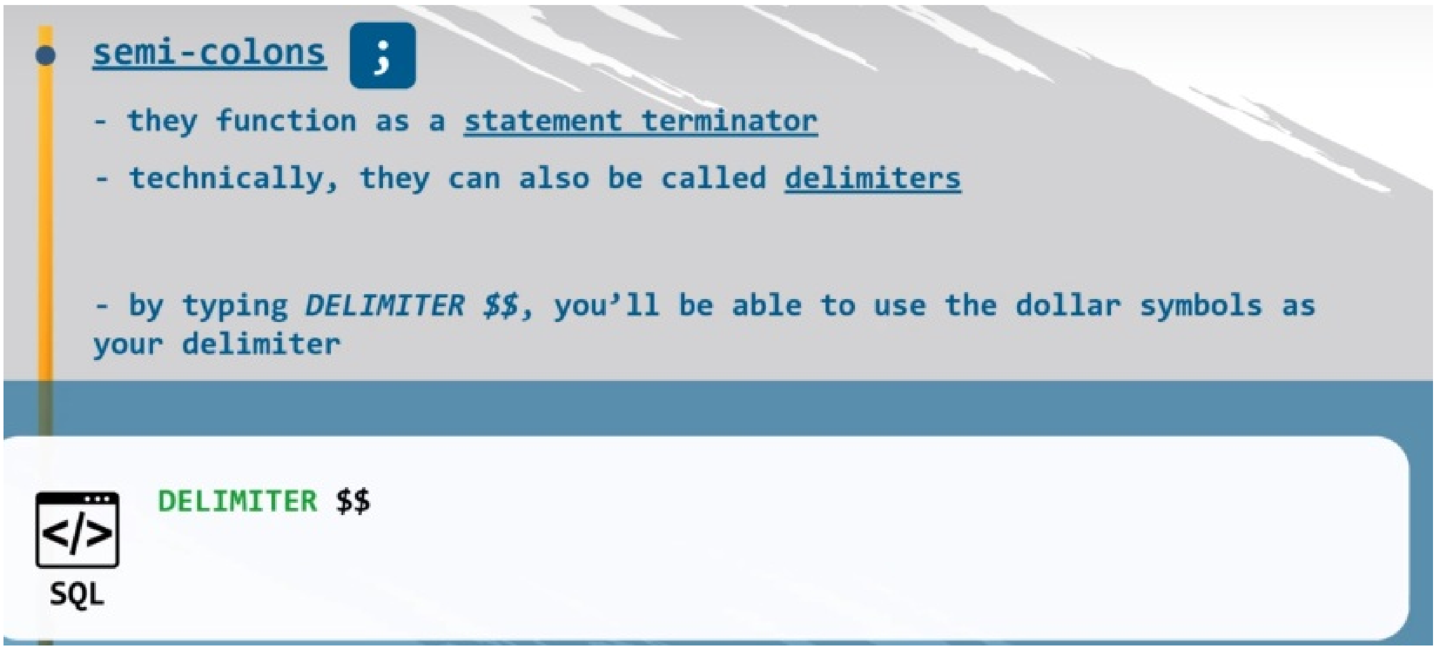
- A stored procedure provides an important layer of security between the user interface and the database. It supports security through data access controls because end users may enter or change data, but do not write procedures.

- A procedure in SQL (often referred to as stored procedure), is a reusable unit that encapsulates the specific business logic of the application. A SQL procedure is a group of SQL statements and logic, compiled and stored together to perform a specific task.

Types Of Stored Procedures.

- Stored Procedure :- Procedures
- Functions :- User Defined Functions and Build in Functions (Aggregate function and datetime functions)

Stored Procedure.



semi-colons ;

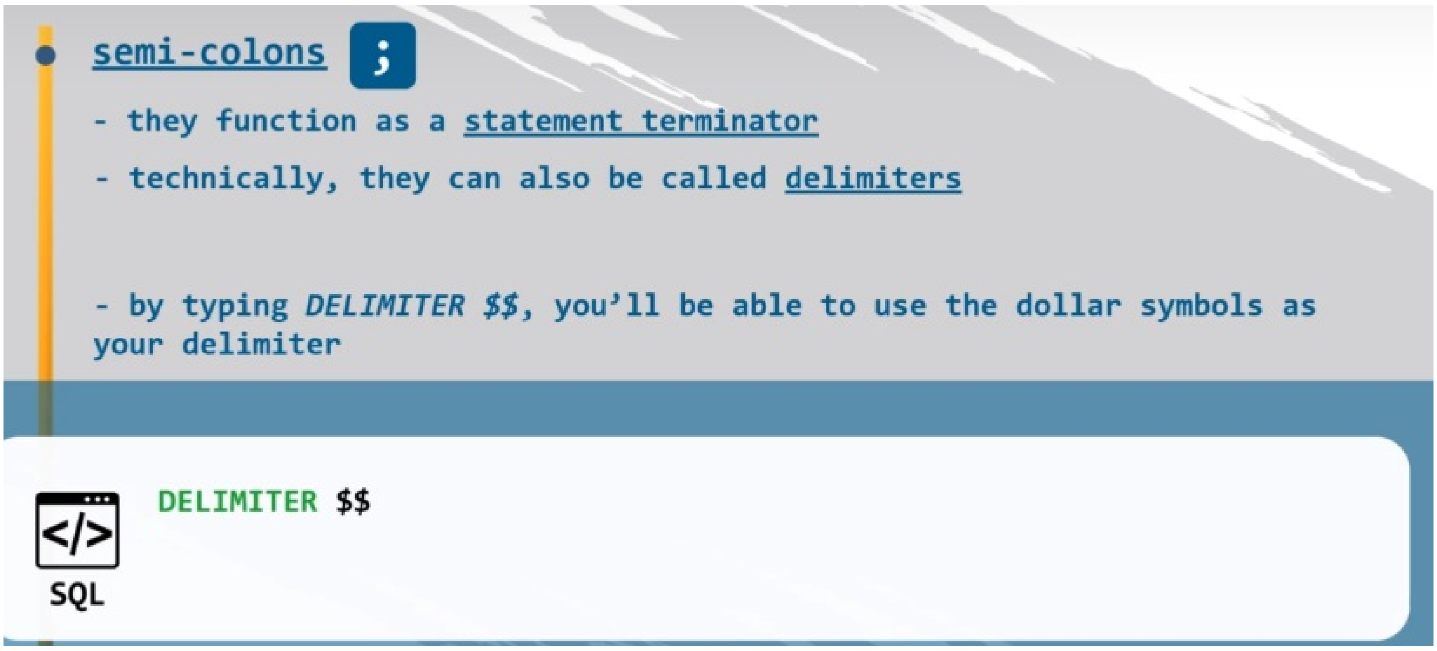
- they function as a statement terminator

- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

In the SQL query,each query terminated by the semicolon symbol.Now imagine if we are goiing to invoking certain procedure that uses the semicolon as delimiter.In this case delimiter is going to use as semicolon inside certain procedure that uses the semicolon as delimiter.In this case delimiter is going to use as semicolon inside stored procedure so it it will run first query and it will not run other queries.


semi-colons ;

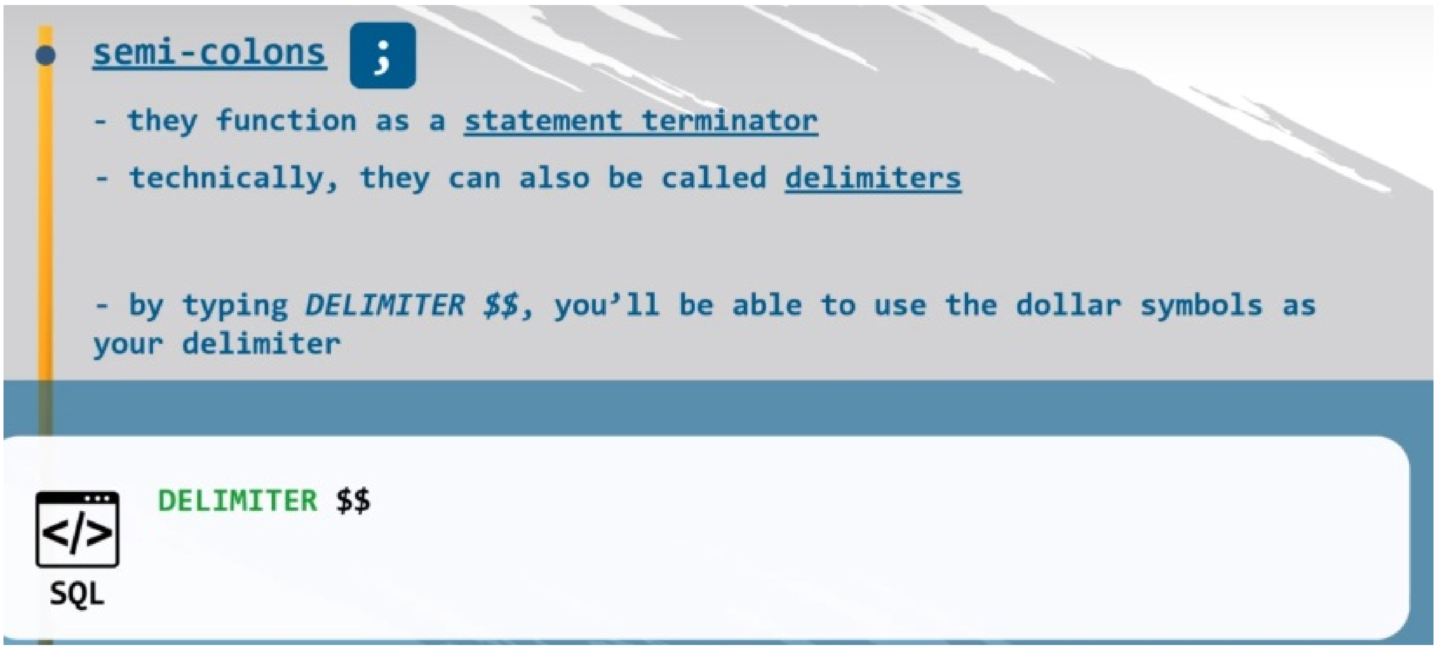
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

In order to avalid this situation.We need temoparary delimiter that will different from standard semicolon.There are many symbol we can use `//` or `$$`.

It does not matter which one you are going to use.For the time being we are going to use the `$` symbol.


semi-colons ;

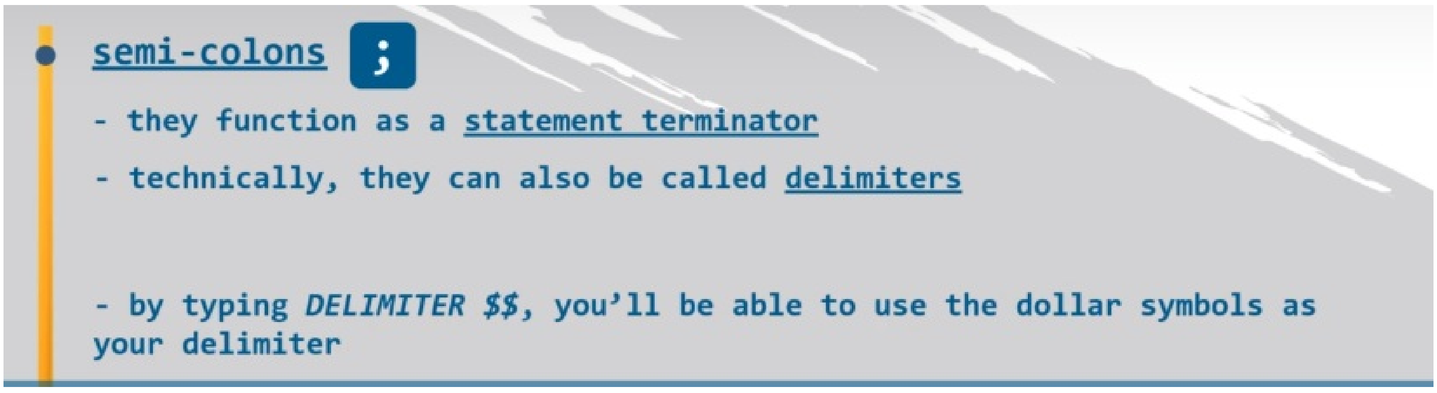
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

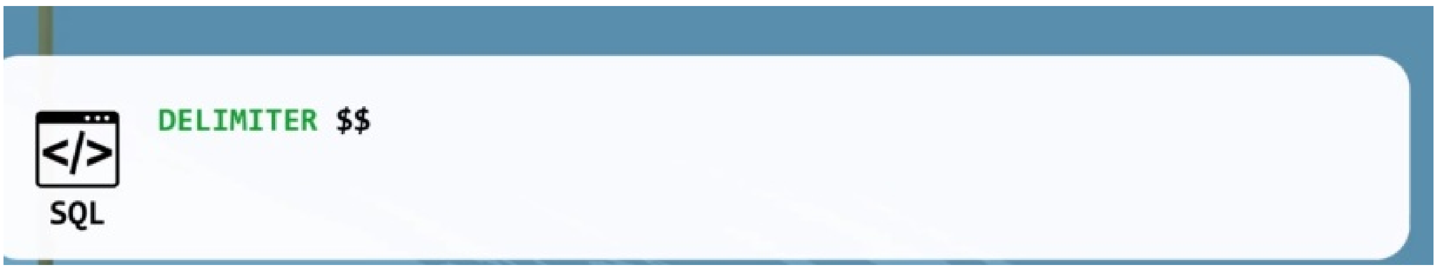
It will allow us to execute the all queries of SQL in Stored Procedure.

Syntax

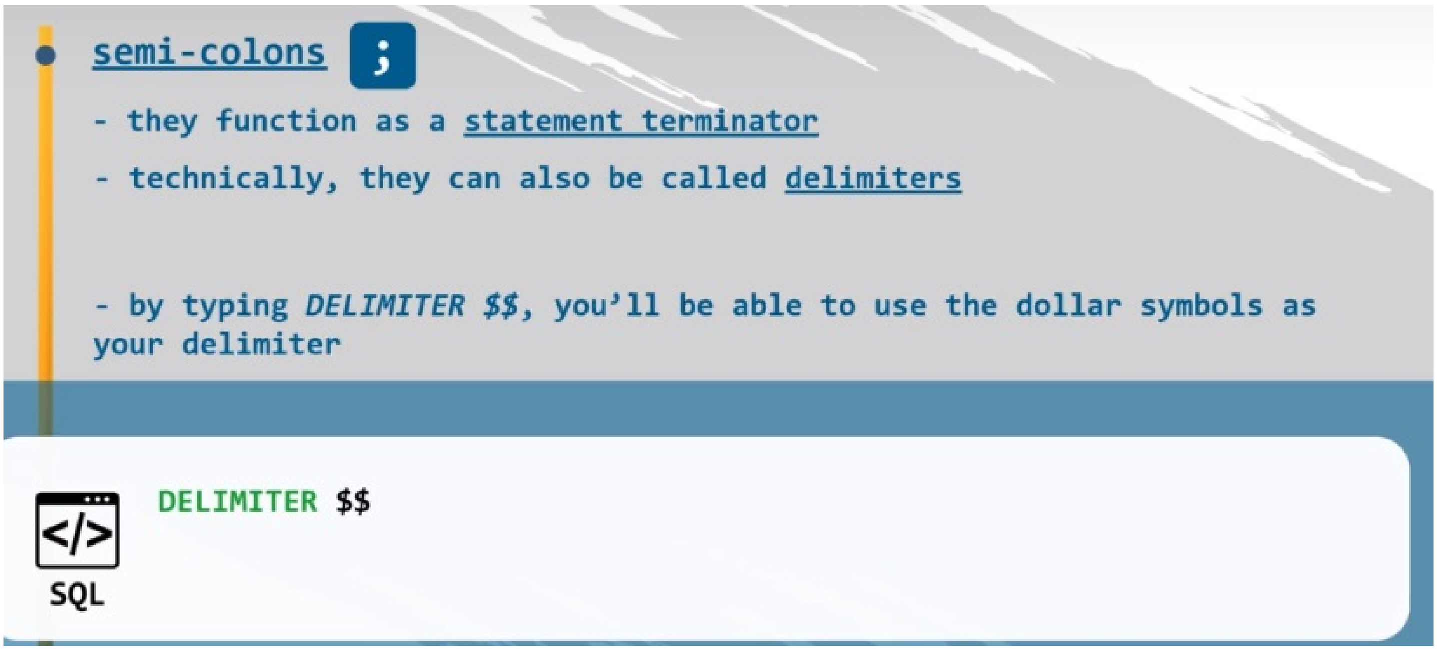

semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter



```sql
DELIMITER $$
```

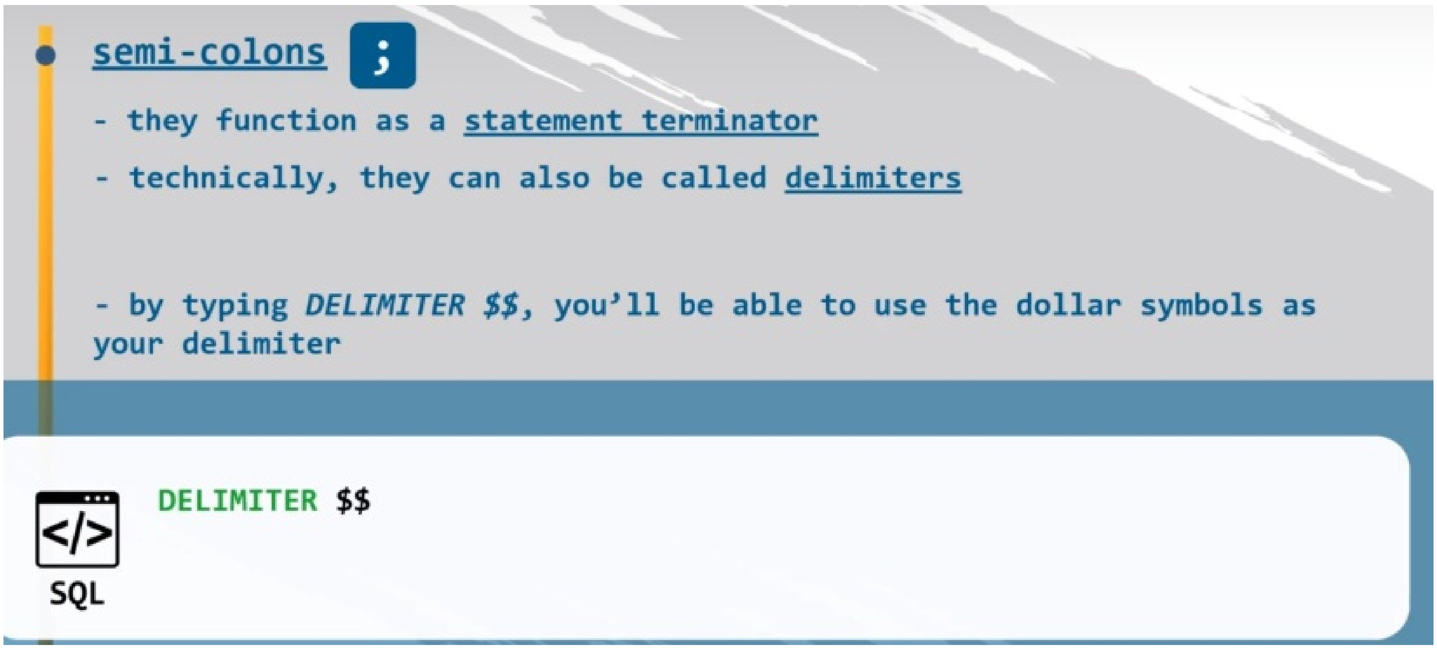


semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```



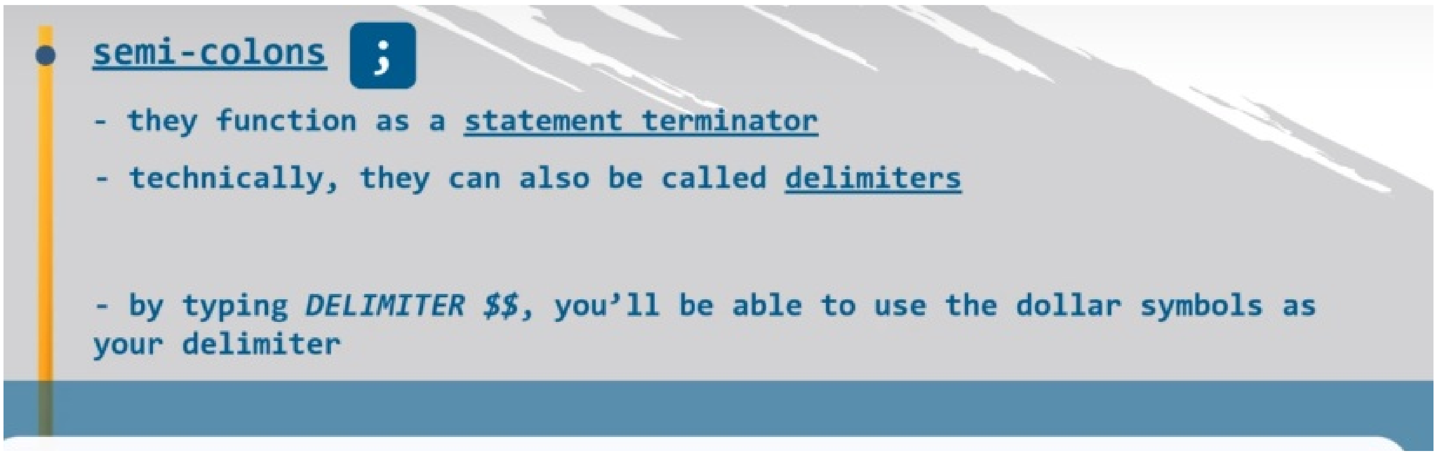
semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

Final Syntax



semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

Non-Parametric Procedure.

Example No. 1

semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

DELIMITER $$

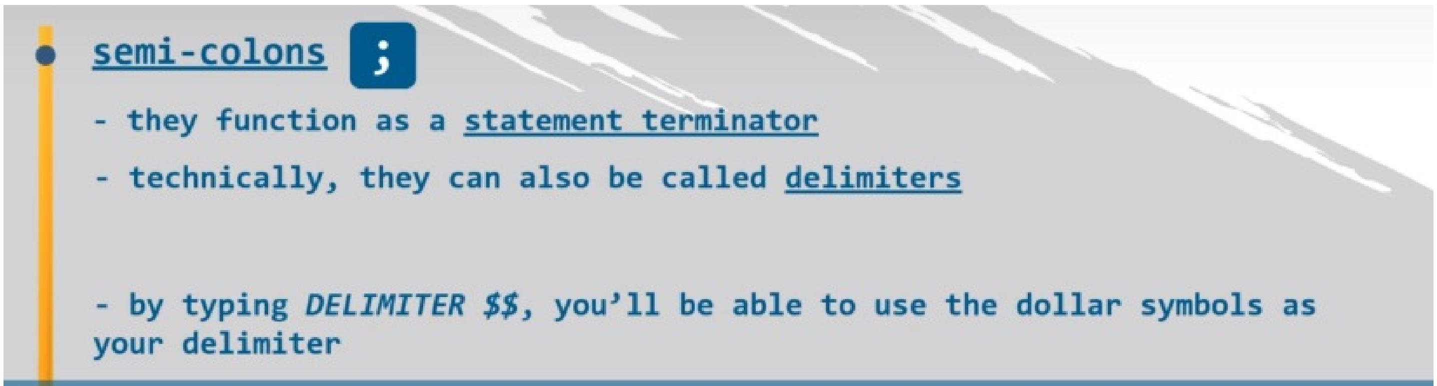
semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

In order to see the logic in the stored procedure we have an option on our left hand side.

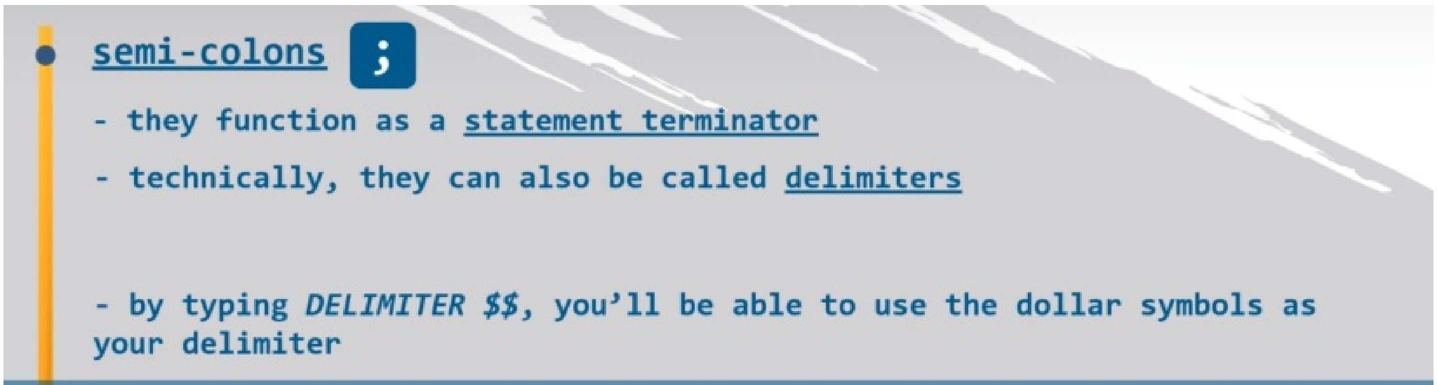

semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

There are three ways of calling the stored procesures


semi-colons ;

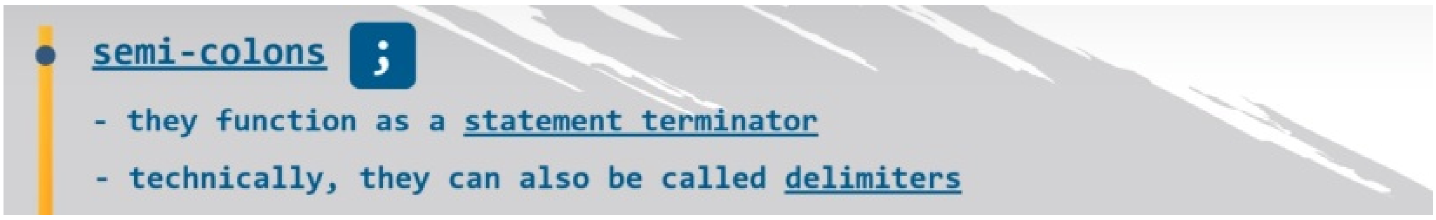
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

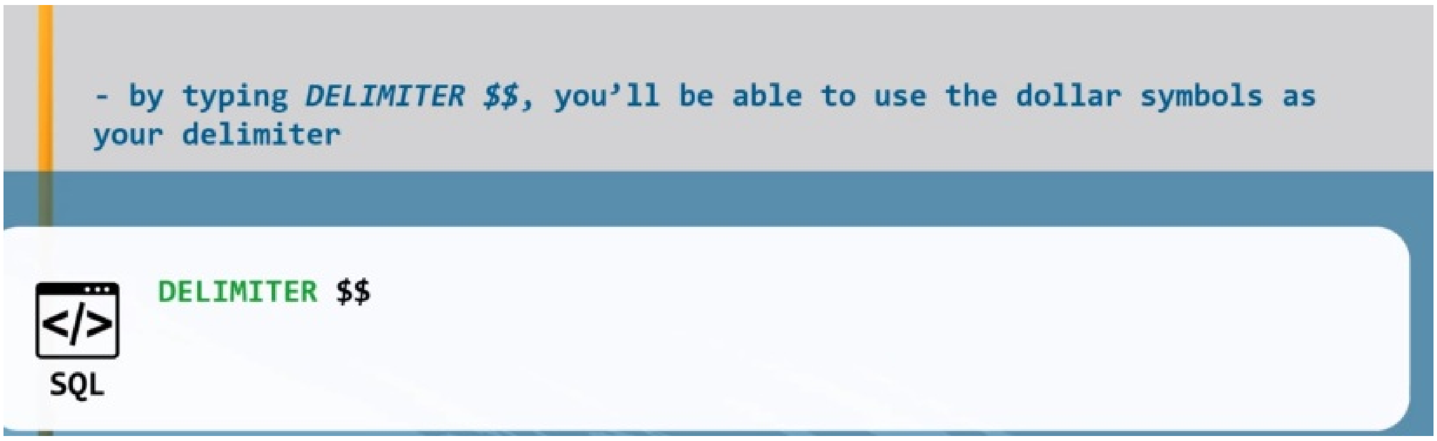
```sql
DELIMITER $$
```

By using an option which is avaliable our left hand side and two are given above.

Another way of making the Stored Procedure.


semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

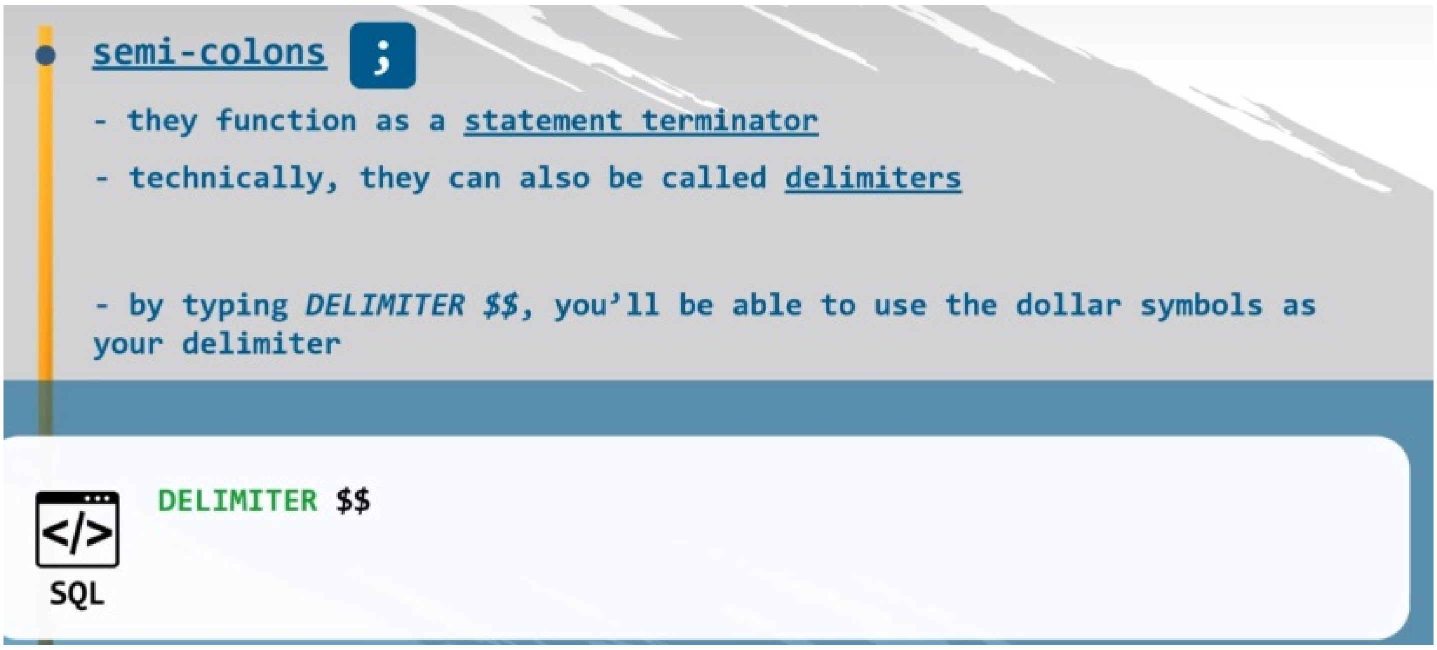


- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```


We just have to click right click on stored procedure and select option that is `new stored procedures.`

Once we click on it will get the code structure of the stored procedure like below.



semi-colons ;

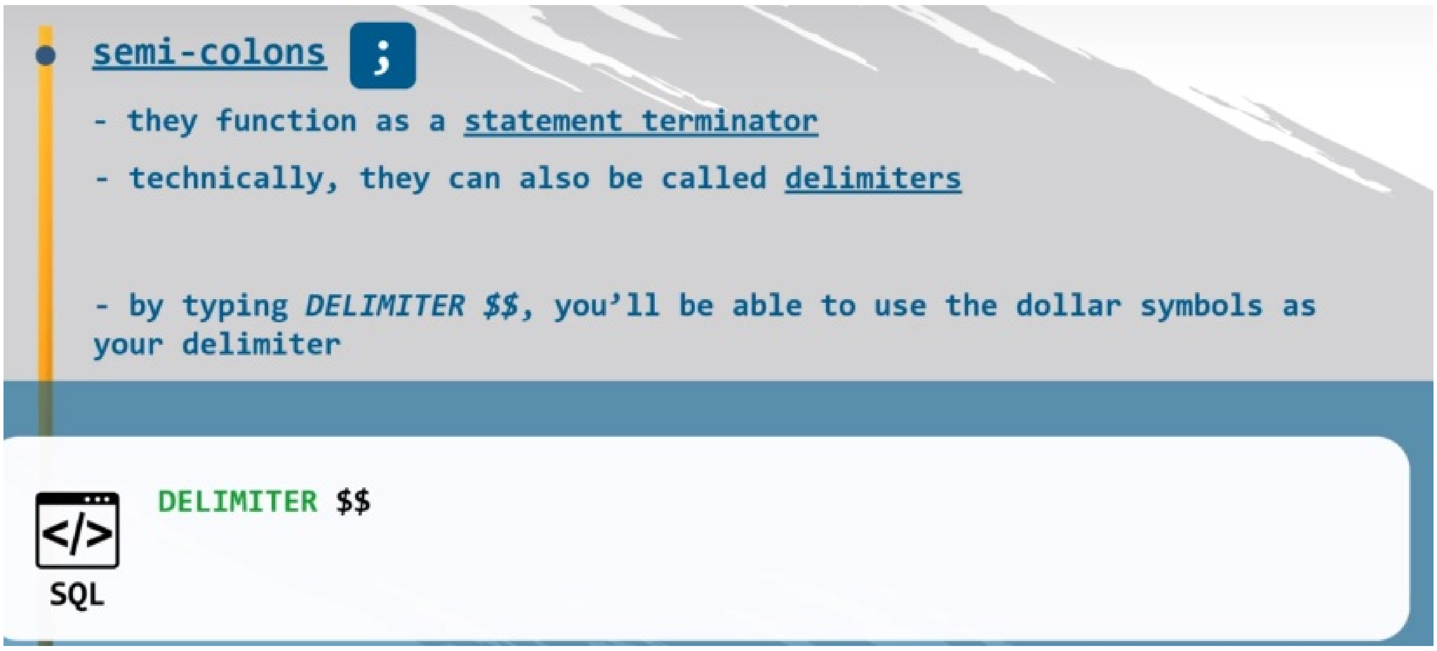
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

We need to Given a proper name to the proocedure and start writing code inside the BEGIN and END block.

Once we done with everything we need to click on apply button.



semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

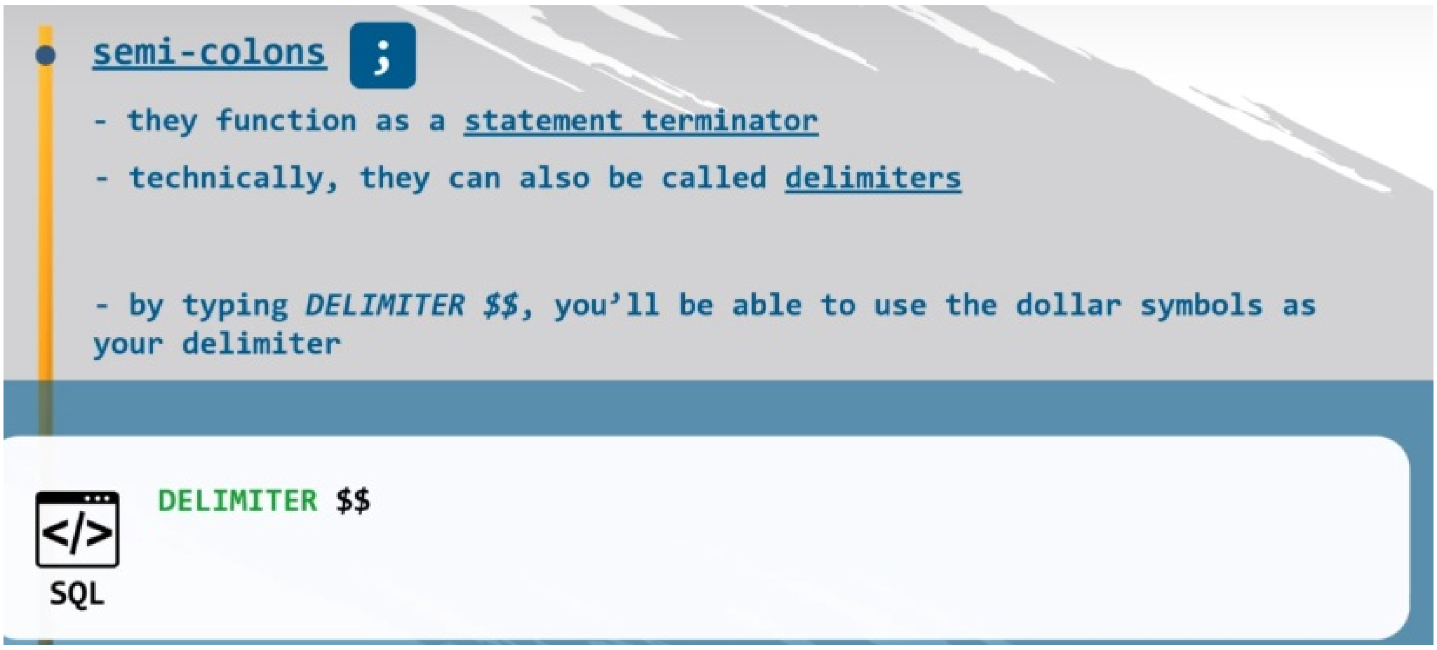
- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

Then we will get the stored procedure at the end.

In order to drop the procedure we can drop it with the help `DROP PROCEDURE PROCEDURE_NAME` and another way to do that with the help of left hand side stored procedure right click on stored procedure and select the `DROP procedure` option.

Like below way :-



semi-colons ;

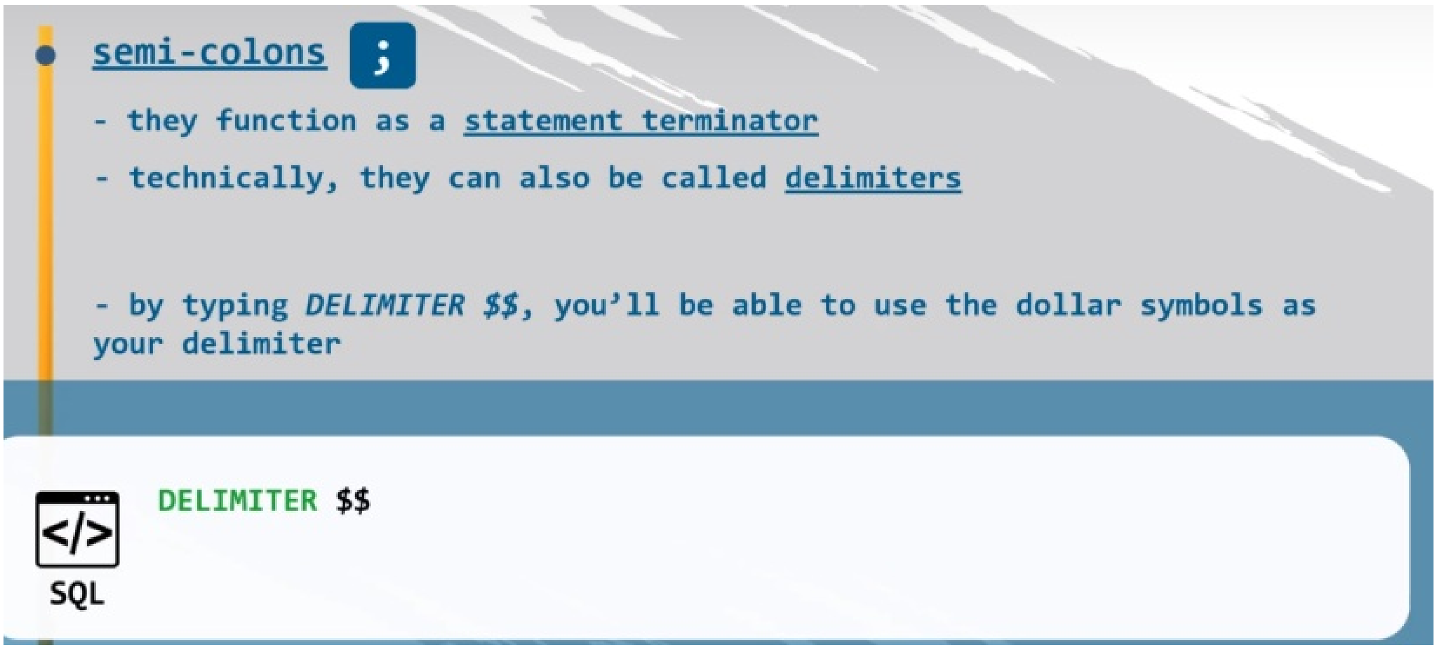
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

Stored Procedure With Input Parameter.

Inside the stored procedure we have to provide the input parameters like below way.



semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

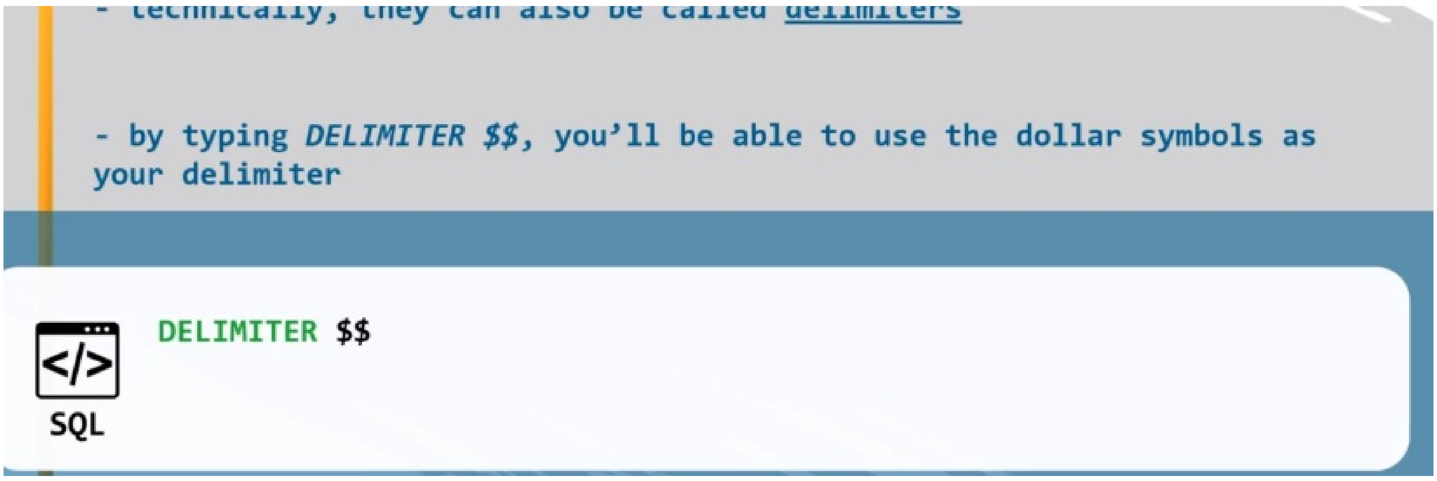
```sql
DELIMITER $$
```

IN Parameter

It is an input parameter and its syntax is given below.

semi-colons ;

- they function as a statement terminator



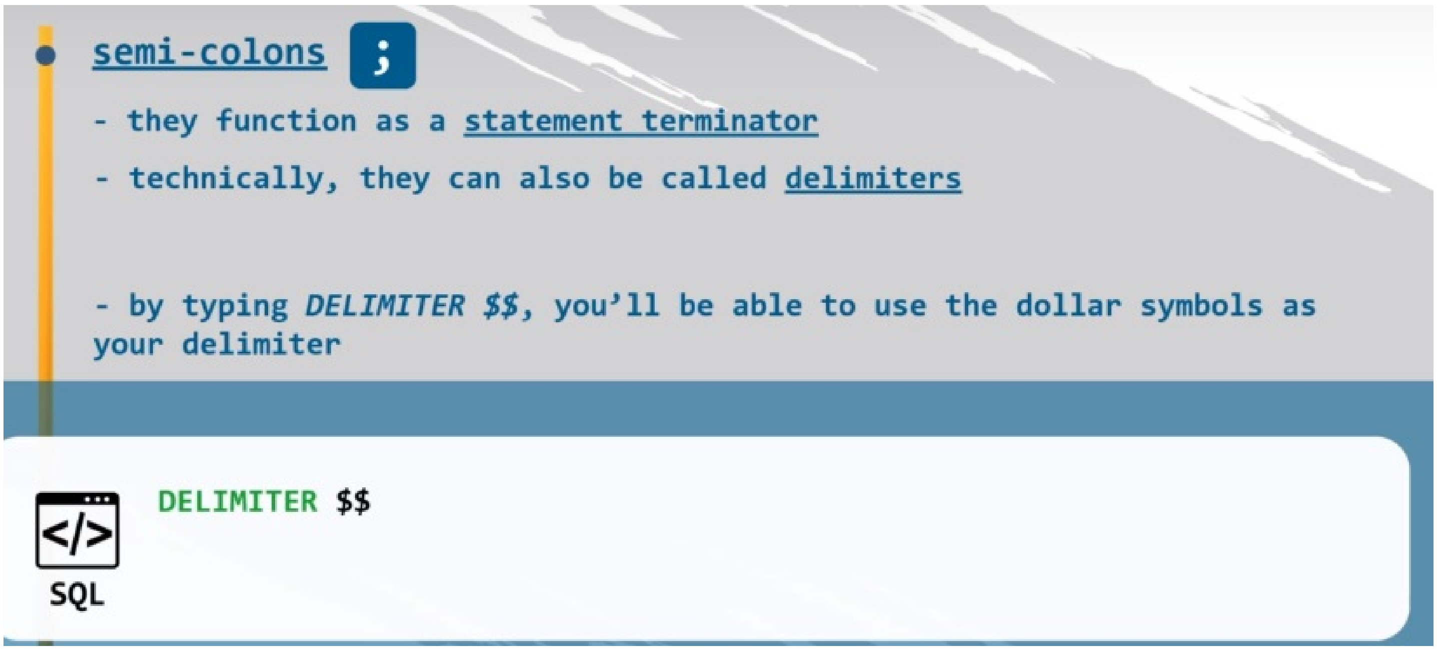
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```


Example No :- 1

Please check the below code.



semi-colons ;

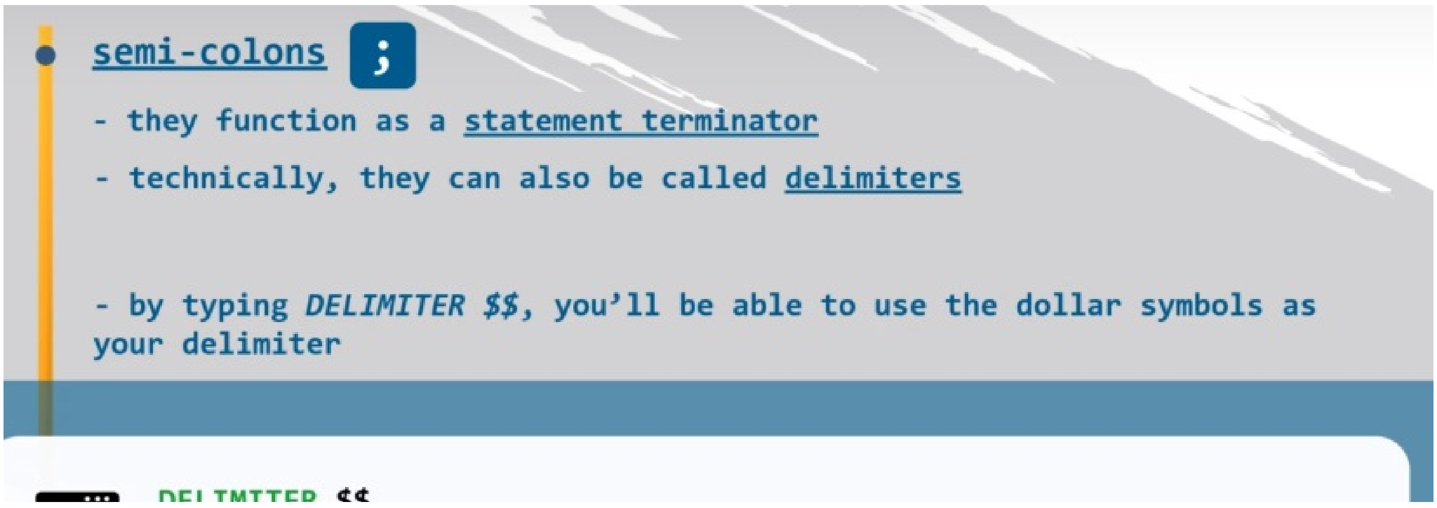
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

- **Here,I am getting the stored procedures example.If we can see `P_film_id` is input parameter with data type of `INTEGER` and we have written code inside the `BEGIN` and `END` keyword.**
- **If we have mention the condition in `WHERE` clause.That if the film_id will equal to the Input parameter then It will return the all the Title,rental_rate,last_update from two joined table.**

For the execution of this query we have to click on the right side on stored procedure and the we will pop up window like below.

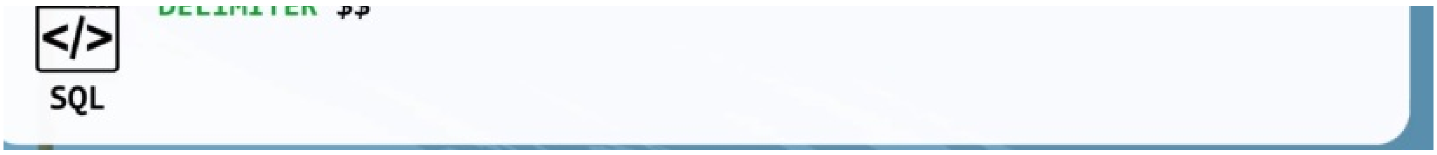


semi-colons ;

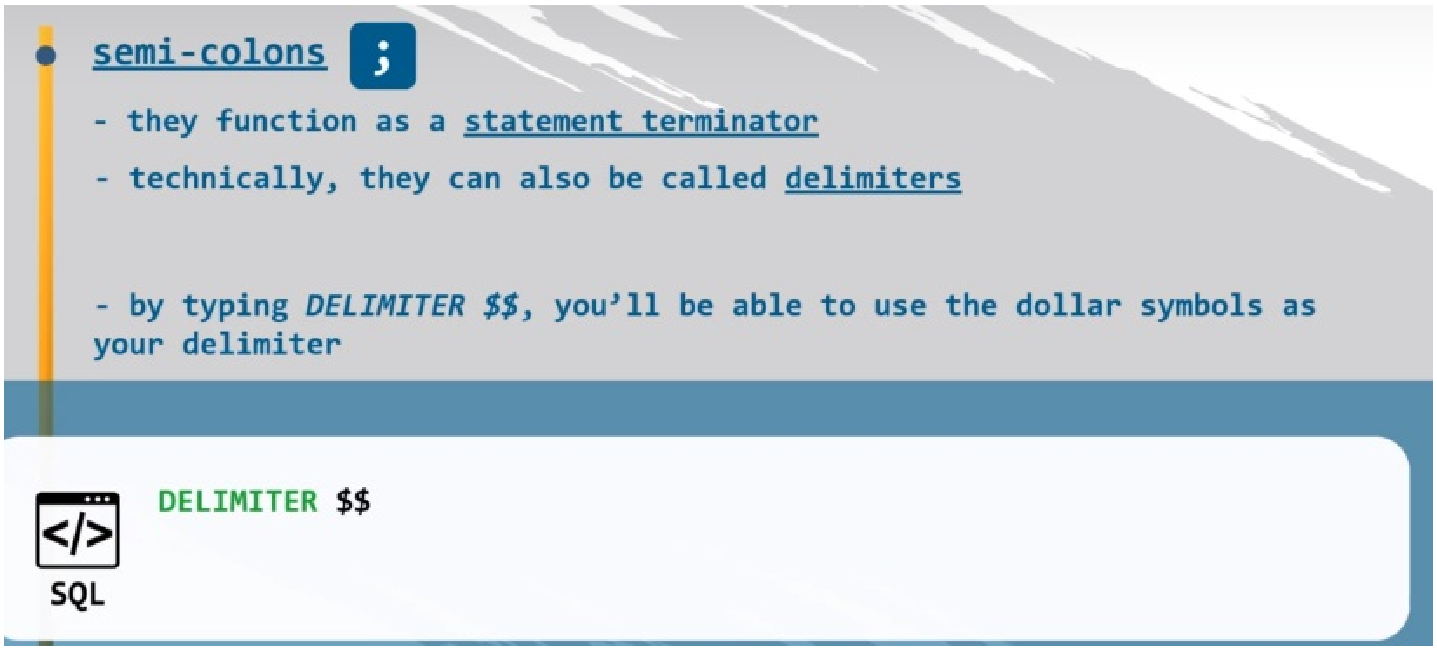
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```


```
DELIMITER $$
```

Once,we have given input parameter inside the that box and click on execute we will get the desirable output like below.

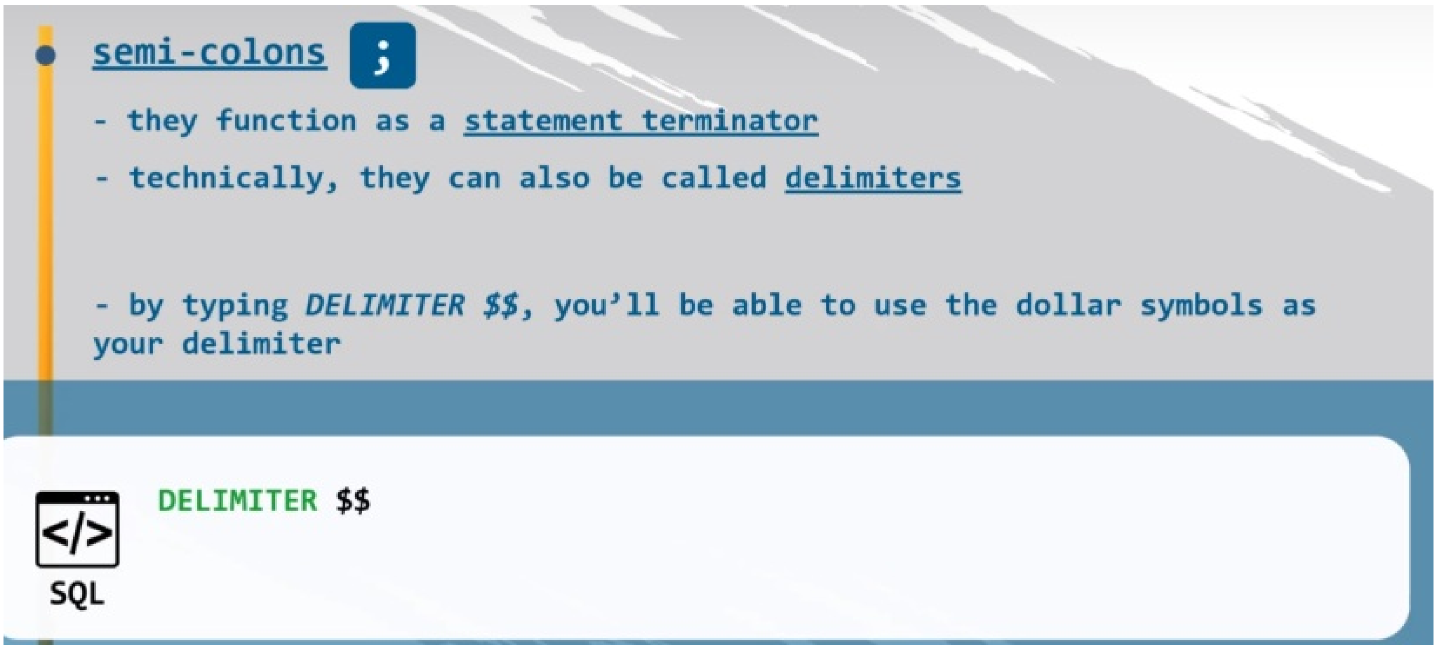

semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```
DELIMITER $$
```

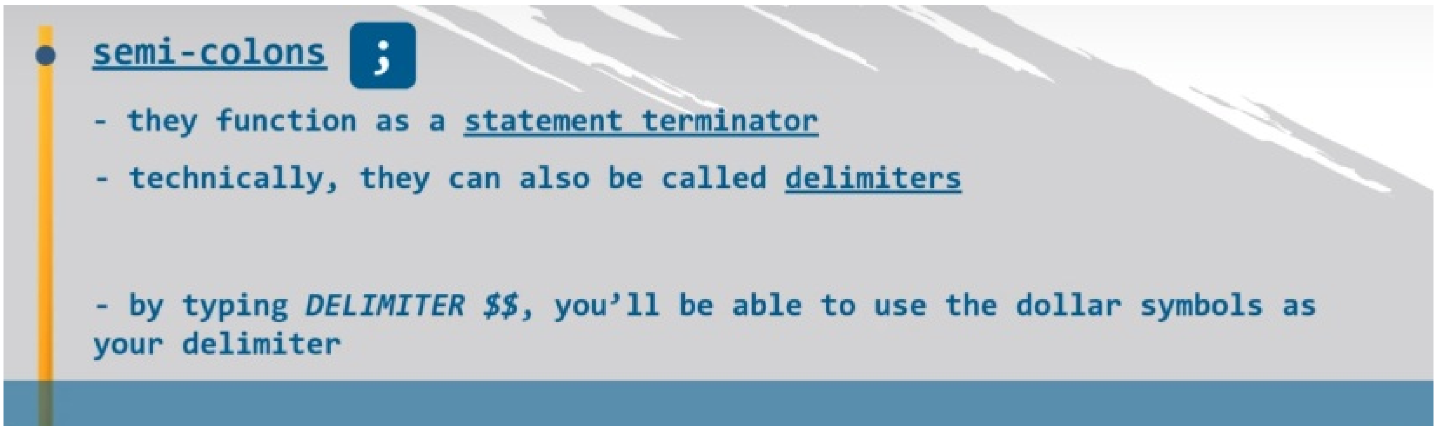

Stored Procedure With Output Parameter.


semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```
DELIMITER $$
```



semi-colons ;

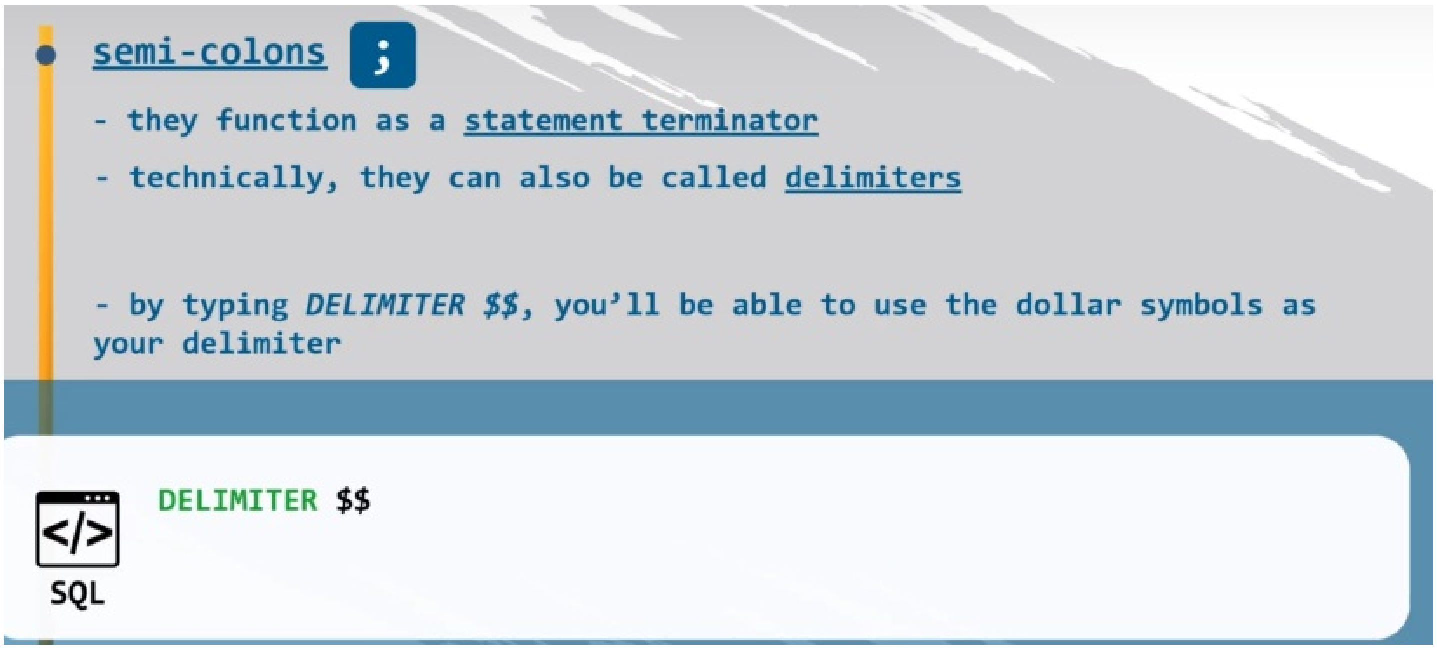
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```
DELIMITER $$
```

In the stored procedure we always have to provide the Input and Output parameter and for insering output parameter we have always use `SELECT..INTO`.

Syntax


semi-colons ;

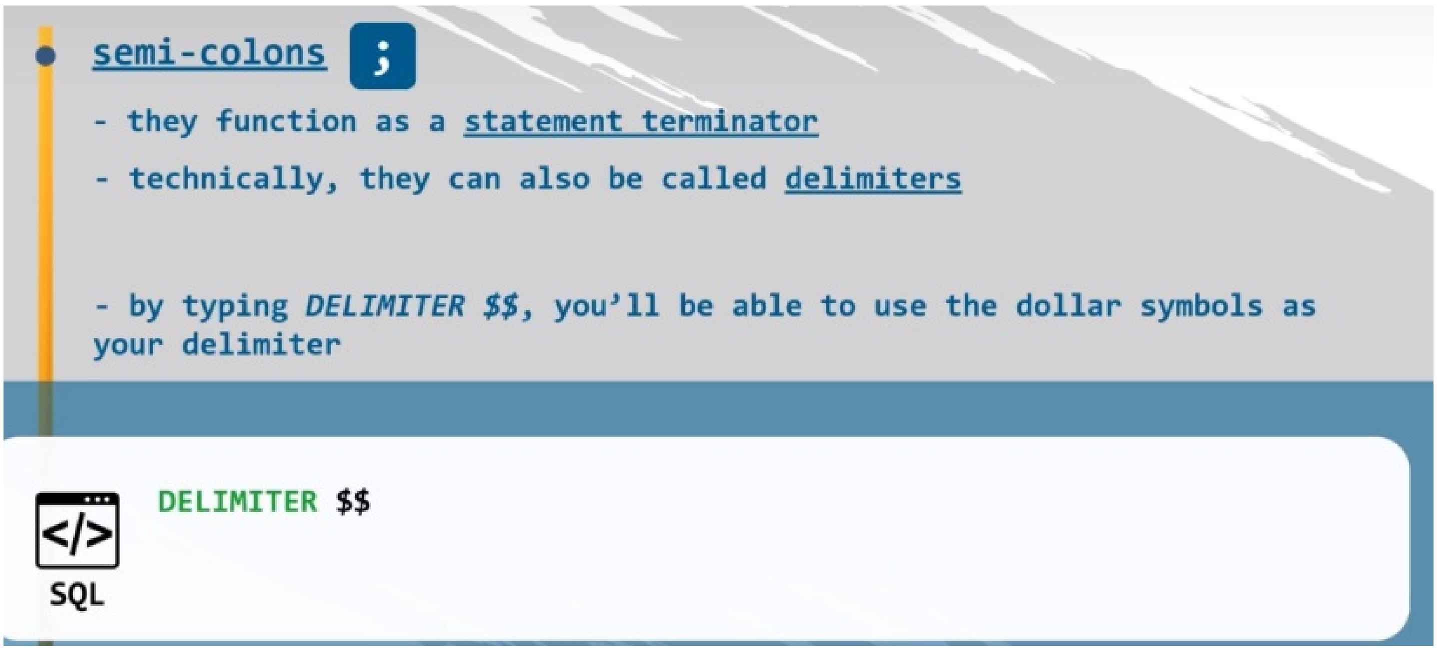
- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```
DELIMITER $$
```

Once we write a code and for the execution we have do the right click on stored procedure name and run that stored procedure.

We will get the pop up window like below way..

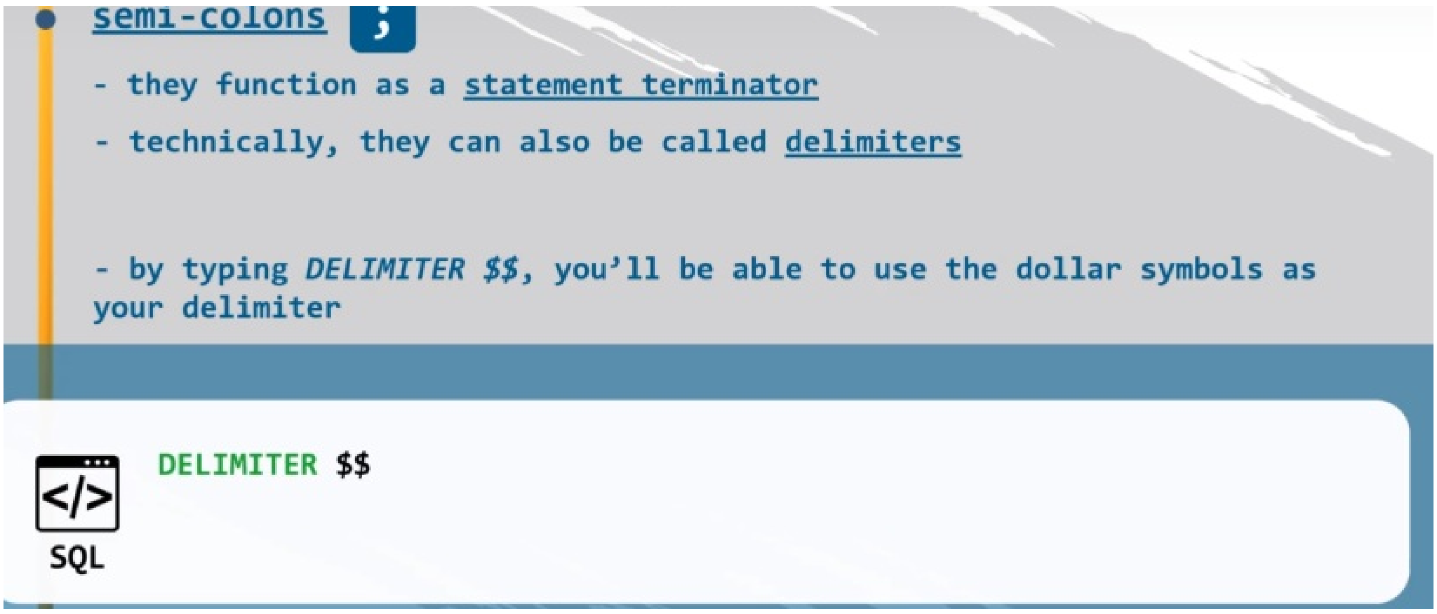

semi-colons ;

- they function as a statement terminator
- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you'll be able to use the dollar symbols as your delimiter

```
DELIMITER $$
```

Once given the input parameter in `P_film_id` as `1` and press Execute button,we will get the output as below way.



semi-colons ;

- they function as a statement terminator

- technically, they can also be called delimiters

- by typing *DELIMITER $$*, you’ll be able to use the dollar symbols as your delimiter

```sql
DELIMITER $$
```

In [ ]:

In [ ]:

In [ ]:

About Views In SQL

- What are Views in MySQL?
- **VIEWS** are virtual tables that do not store any data of their own but display data stored in other tables.
- In other words, VIEWS are nothing but SQL Queries. A view can contain all or a few rows from a table.
- A MySQL view can show data from one table or many tables.

Why We Use MySQL VIEWS ?

- Because MySQL views look and function like regular tables, they are sometimes called virtual tables.
- Views offer a number of advantages.
- You can use views to hide table columns from users by granting them access to the view and not to the table itself.
- This helps enhance database security and integrity.

Are VIEW Faster than Query ?

- Views make queries faster to write, but they don't improve the underlying query performance.
- Once we create an indexed view, every time we modify data in the underlying tables then not only must SQL Server maintain the index entries on those tables, but also the index entries on the view.

- No, a view is simply a stored text query. You can apply WHERE and ORDER against it, the execution plan will be calculated with those clauses taken into consideration.

Does MySQL Views Improve the performance ?

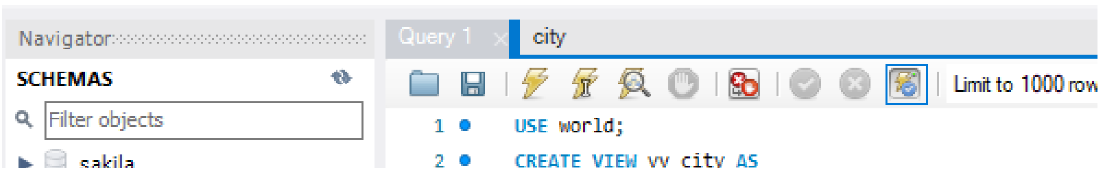
- It totally depends on what you are viewing through view.
- But most probably reducing your effort and giving higher performance. When SQL statement references a nonindexed view, the parser and query optimizer analyze the source of both the SQL statement and the view and then resolve them into a single execution plan

Some More Information about The Views.

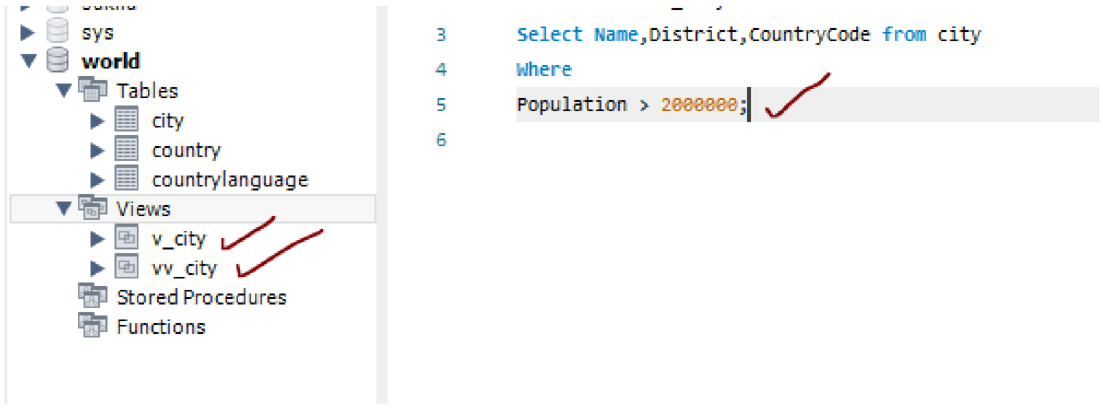
SQL Views

- It is an vertual table whose contents are obtained from existing table or tables,called as based table.
- The retrieval happens throgh an SQL statement,incorporated into the view.
- Think of a view object as a view into the base table.
- The view itself does not contain any real data;data is physically stored in the base table.
- The Views simply shows the data contained in base table.

Example 1:-

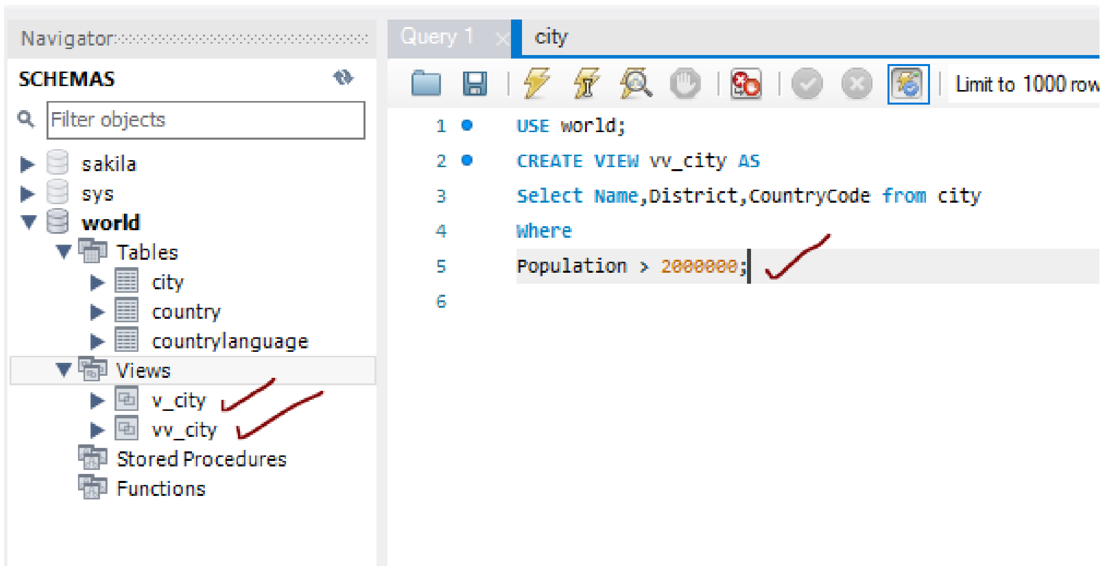


```
USE world;
CREATE VIEW vv city AS
```



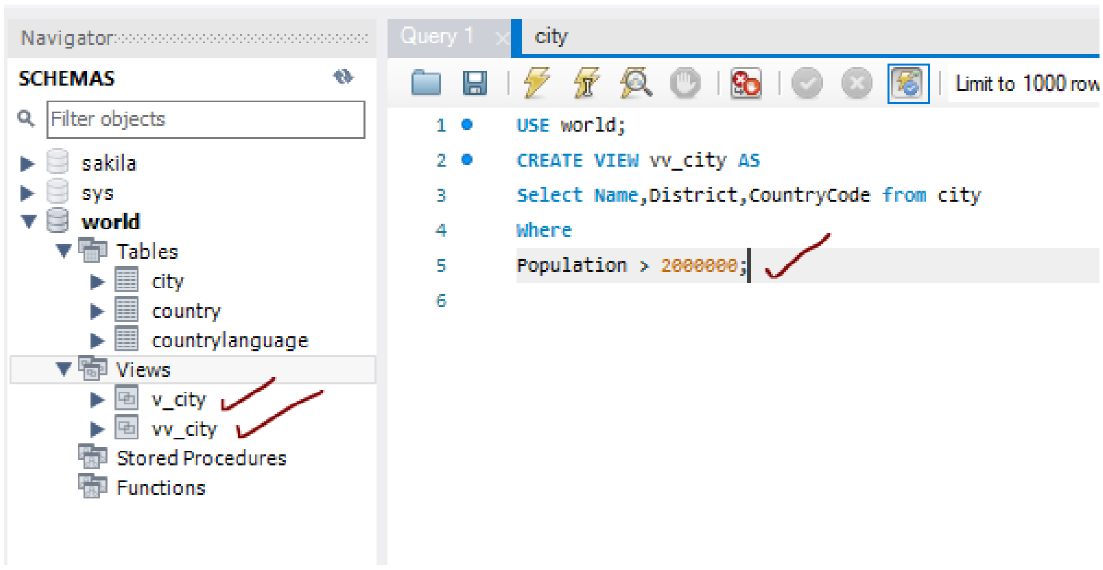
```
Select Name,District,CountryCode from city
Where
Population > 2000000;
```

Example 2:-



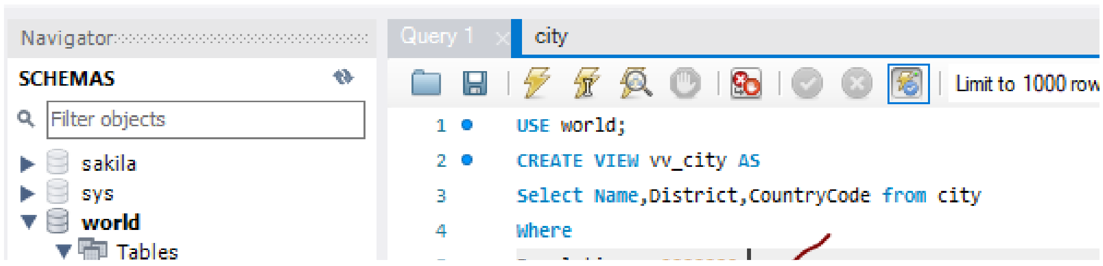
```
USE world;
CREATE VIEW vv_city AS
Select Name,District,CountryCode from city
Where
Population > 2000000;
```

Example 3:-

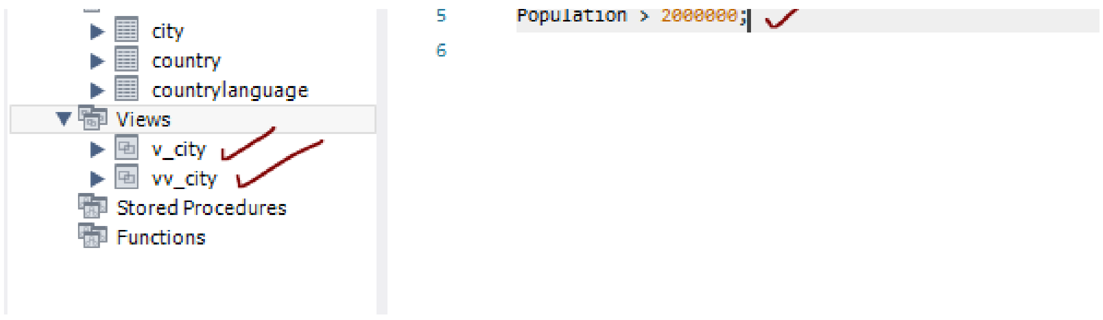


```
USE world;
CREATE VIEW vv_city AS
Select Name,District,CountryCode from city
Where
Population > 2000000;
```

Example 4 :-



```
USE world;
CREATE VIEW vv_city AS
Select Name,District,CountryCode from city
Where
```

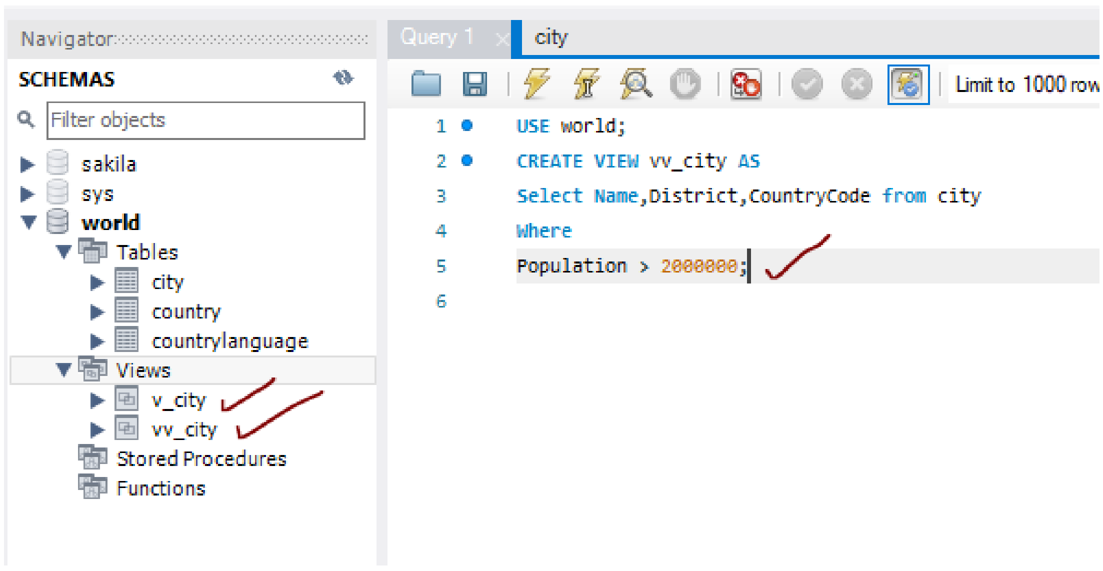


```
Population > 2000000;
```

Important Notes

- **In that we can see we have multiple user to see the specific information,for that we have write code and execute the query for all the user but it seems like tedious process and it will take lot of time.**

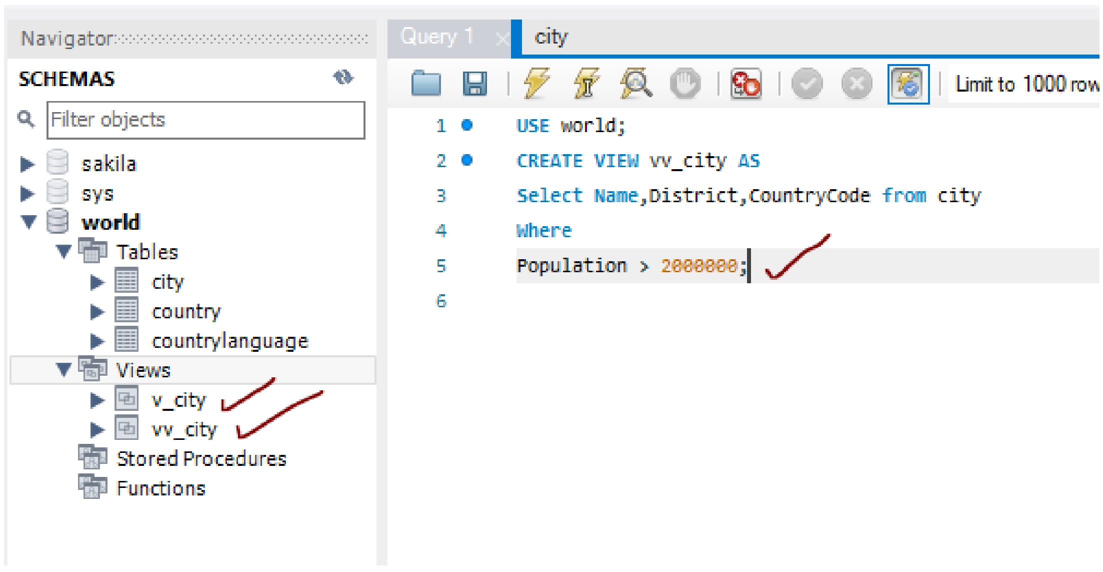
- **If you write the Query with VIEW function then the view act as shortcut for the same SELECT Statement every time new request has been made.**



```
USE world;
CREATE VIEW vv_city AS
Select Name,District,CountryCode from city
Where
Population > 2000000;
```

Advantages.

- **VIEW save lot of coding time.**
- **Accupying new extra memory.**
- **SQL VIEW act as dynamic table because its instantly reflects data and structural changes in the base table.**



```
USE world;
CREATE VIEW vv_city AS
Select Name,District,CountryCode from city
Where
Population > 2000000;
```

Thank You !!