Fusion Software Institute Pune

SQL
Structured Query Language

Installation Link

- https://dev.mysql.com/downloads/windows/installer/8.0.html

Some settings

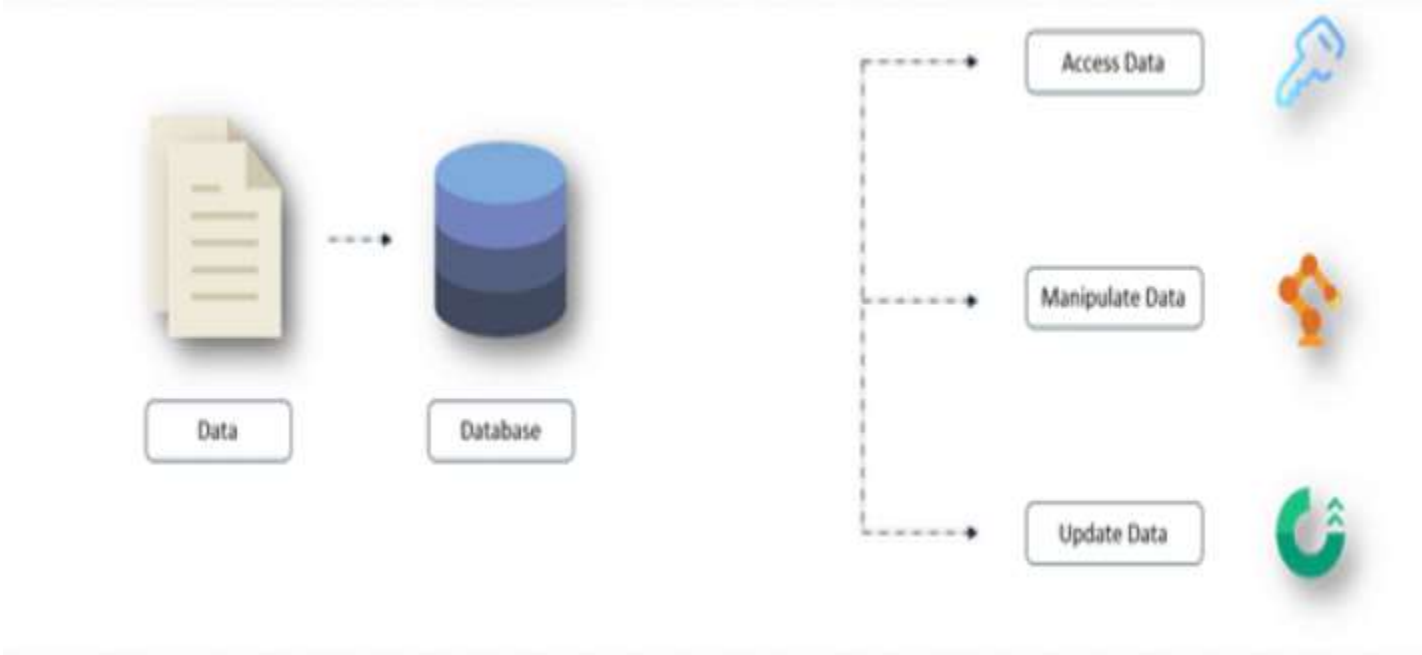
- **How to set default database in MySQL Workbench**
- Go to the database-> Manage Connection -> Default Schema : test[put the database name] -> click on Test Connection

- **How to change Font in MySQL Workbench**
- Edit->preferences-> Appearance
- First row in font column double click [change no eg 19]

- **Setting for update operation**
- Edit-> Preferences -> SQL Editor-> Uncheck the Safe Updates checkbox

What Is a Database?

- The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently. It is also used to organize the data in the form of a table, schema, views, and reports, etc.

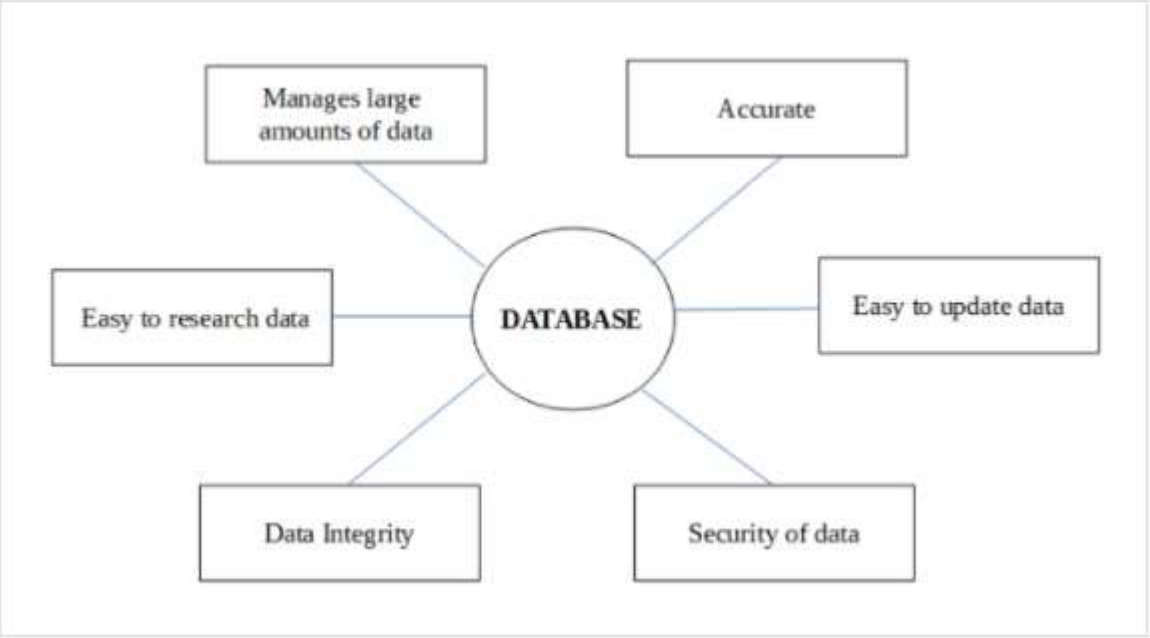


Example of Database

- 1) The college Database organizes the data about the admin, staff, students and faculty etc. Using the database, you can easily retrieve, insert, and delete the information.
- 2) An online telephone directory uses a database to store data of people, phone numbers, and other contact details. Your electricity service provider uses a database to manage billing, client-related issues, handle fault data, etc.
- 3) Let us also consider Facebook. It needs to store, manipulate, and present data related to members, their friends, member activities, messages, advertisements, and a lot more. We can provide a countless number of examples for the usage of databases.

Why do we need database

- A database is a collection of data, usually stored in electronic form. A database is typically designed so that it is easy to store and access information.
- A good database is crucial to any company or organisation. This is because the database stores all the details about the company such as employee records, transactional records, salary details etc.



- The various reasons a database is important are −

1) Manages large amounts of data

- A database stores and manages a large amount of data on a daily basis. This would not be possible using any other tool such as a spreadsheet as they would simply not work.

2) Accurate

- A database is pretty accurate as it has all sorts of build in constraints, checks etc. This means that the information available in a database is guaranteed to be correct in most cases.

3) Easy to update data

- In a database, it is easy to update data using various Data Manipulation languages (DML) available. One of these languages is SQL.

4) Security of data

- Databases have various methods to ensure security of data. There are user logins required before accessing a database and various access specifiers. These allow only authorised users to access the database.

5) Data integrity

- This is ensured in databases by using various constraints for data. Data integrity in databases makes sure that the data is accurate and consistent in a database.

6) Easy to research data

- It is very easy to access and research data in a database. This is done using Data Query Languages (DQL) which allow searching of any data in the database and performing computations on it.

Types of Databases

Relational

Data stored in tables

Non-relational (NoSQL)

data not stored in tables

** We use SQL to work with relational DBMS

SQL - RDBMS Concepts

- RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- A relational database is a collection of information that organizes data in predefined relationships where data is stored in one or more tables (or "relations") of columns and rows, making it easy to see and understand how different data structures relate to each other. Relationships are a logical connection between different tables, established on the basis of interaction among these tables.

• What is a table?

- The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows.
- Remember, a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

- **What is a field?**
- Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.
- A field is a column in a table that is designed to maintain specific information about every record in the table.
- **What is a Record or a Row?**
- A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table –

```
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

- A record is a horizontal entity in a table.

- **What is a column?**
- A column is a vertical entity in a table that contains all information associated with a specific field in a table.
- For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below –

```
+-----------+
| ADDRESS   |
+-----------+
| Ahmedabad |
| Delhi     |
| Kota      |
| Mumbai    |
| Bhopal    |
| MP        |
| Indore    |
+----+------+
```

- **What is a NULL value?**
- A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.
- It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

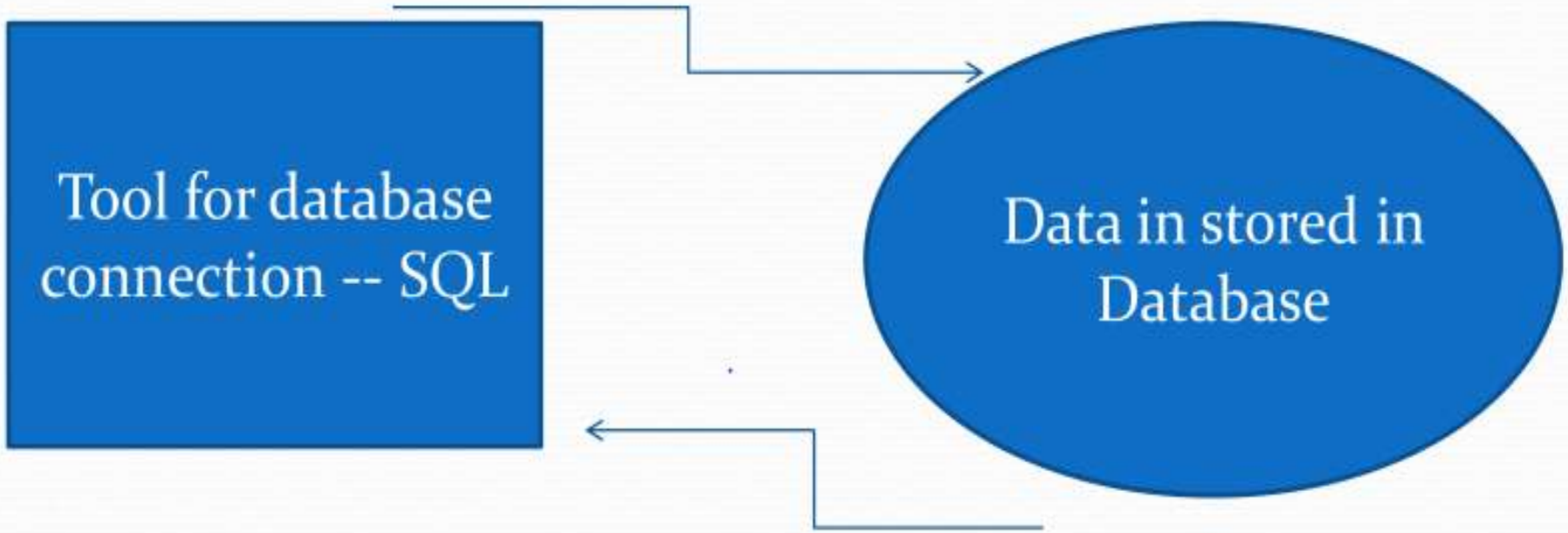
What is SQL

- SQL is a programming language used to interact with relational database
- SQL is a standard language for accessing and manipulating databases.
- SQL lets you access and manipulate databases
- SQL keywords are NOT case sensitive Ex. SELECT as select
- Semicolon is the standard way to separate each SQL statement in database systems. ex. Select * from employee;
- It is Standard language for dealing with Relational Database which can be used to Create, Read, Update and Delete database records(CRUD Operations)

What SQL can do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Creating DB, Creating table, updating data in table, Delete

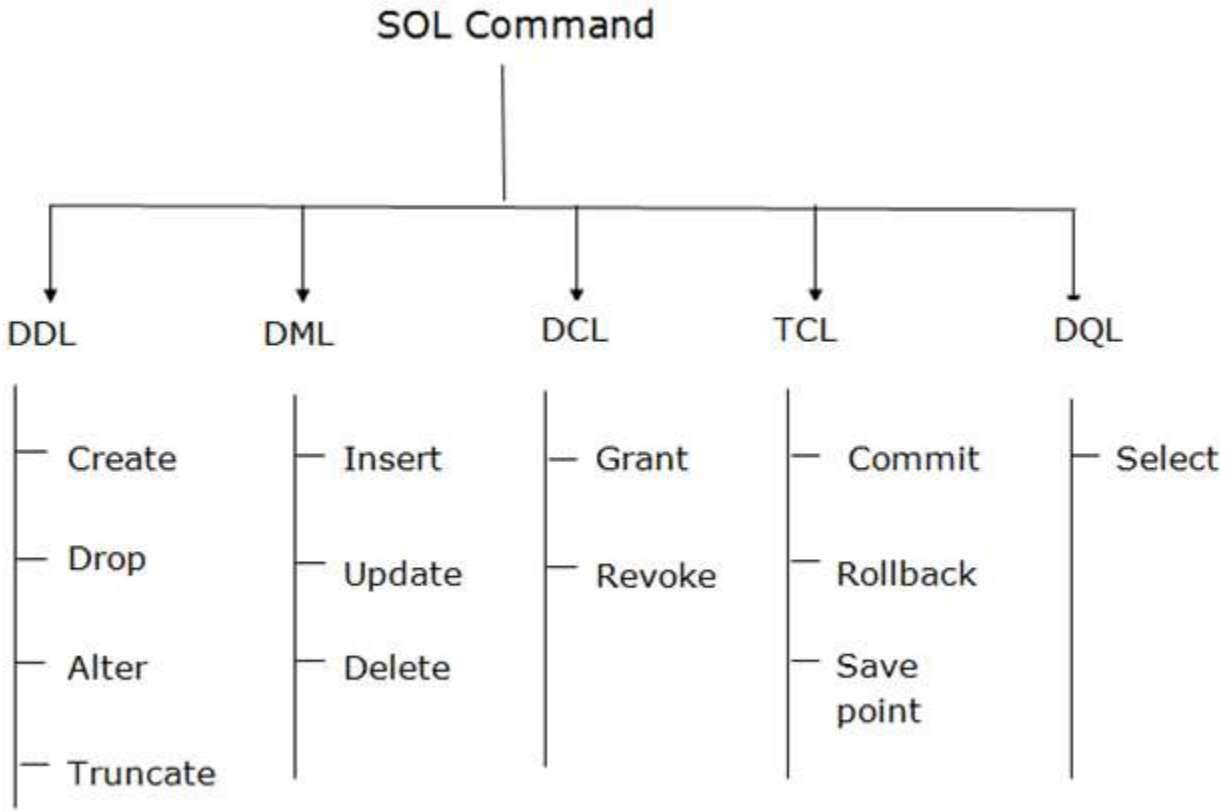


Retrieve data from db

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.
- There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.

Types of SQL Commands



1) Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.
- Here are some commands that come under DDL:
- CREATE
- ALTER
- DROP
- TRUNCATE

- **CREATE** It is used to create a new table in the database.
- **DROP:** It is used to delete both the structure and record stored in the table.
- **ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.
- **TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.

Command	What it does?
CREATE TABLE	It creates new table
DROP TABLE	It deletes the ENTIRE table
ALTER TABLE	Modifies the existing table
TRUNCATE TABLE	deletes the data inside a table, but not the table itself

2. Data Manipulation Language(DML)

- Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data. It is responsible for performing all types of data modification in a database.
- Here are some important DML commands in SQL:
- **INSERT:**
- This is a statement is a SQL query. This command is used to insert data into the row of a table.
- **UPDATE:**
- This command is used to update or modify the value of a column in the table.
- **DELETE:**
- This command is used to remove one or more rows from a table.

3. Data Control Language(DCL)

- DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give “rights & permissions.” Other permission controls parameters of the database system.
- **Examples of DCL commands:**
- Commands that come under DCL:
- **Grant:**
- **Grant/give permission to user.** This command is use to give user access privileges to a database.
- **Revoke:**
- **Take back permission from user**. It is useful to back permissions from the user.

4. Transaction control language(TCL)

- Transaction control language or TCL commands deal with the transaction within the database.
- **Commit:**
- **To save the data permanently.** This command is used to save all the transactions to the database.
- **Rollback: To undo the changes**. Rollback command is used to undo transactions that have not already been saved to the database.
- **Savepoint: To save the data temporarily.** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

5. Data Query Language(DQL)

- Data Query Language (DQL) is used to fetch the data from the database. It uses only one command:
- **SELECT:**
- This command helps you to select the attribute based on the condition described by the WHERE clause.

Different Data Types in SQL

- Data type in SQL basically defines the kind of data that will go into a particular column. All entries of one particular column will be of the same data type.

e_id	e_name	e_salary	e_age	e_gender	e_dept
1	Sam	95000	45	Male	Operations
2	Bob	80000	21	Male	Support
3	Anne	125000	25	Female	Analytics
4	Julia	73000	30	Female	Analytics
5	Matt	159000	33	Male	Sales
6	Jeff	112000	27	Male	Operations

- **Numeric Data Types in SQL:** Numeric data types store all numerical values or integer values.

DATA TYPE	FROM	TO
Bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	-2,147,483,648	2,147,483,647
Smallint	-32,768	32,767
Tinyint	0	255
Bit	0	1
Decimal	-10^38 +1	10^38 -1

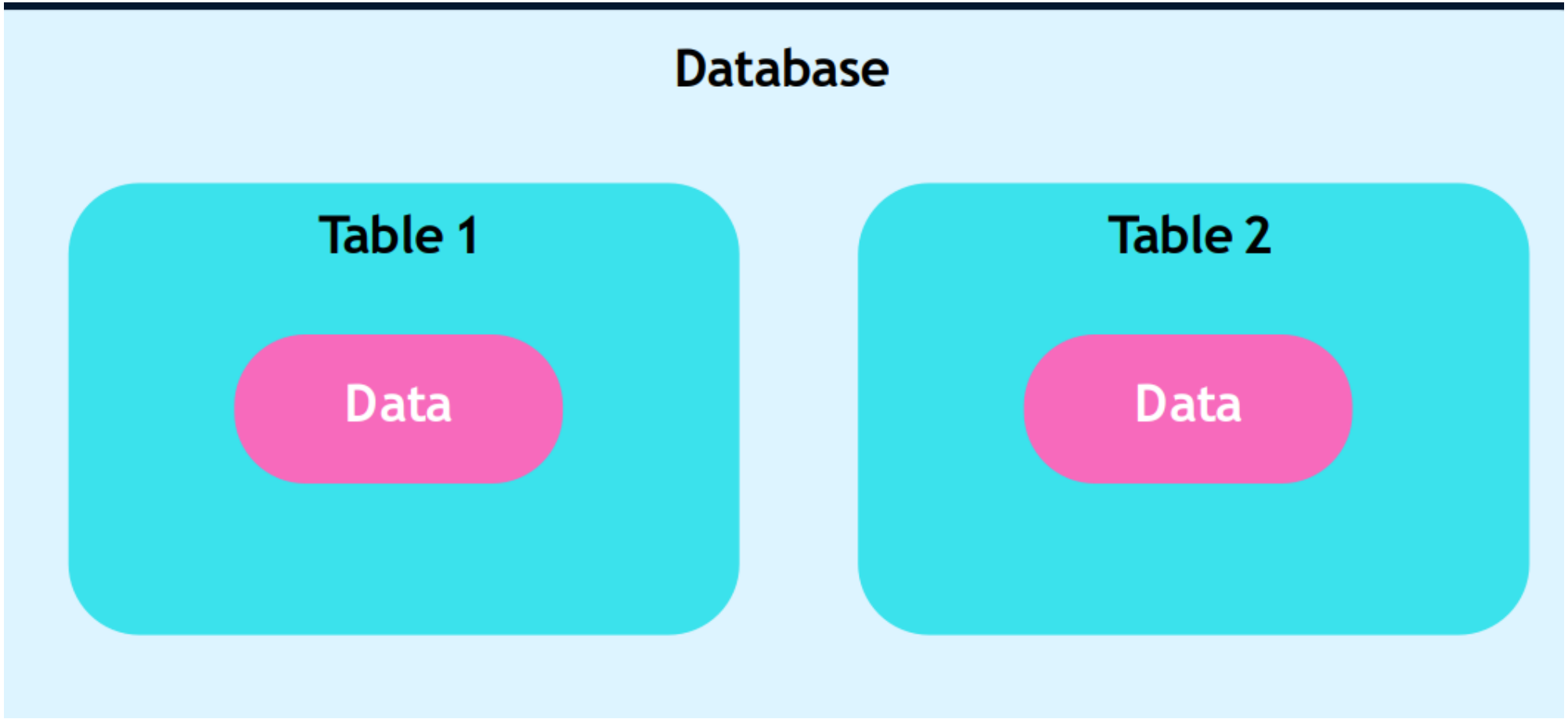
- **Character Data Types in SQL**: Character data types store all alphabetic values and special characters.

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them

- **Date and Time Data Types in SQL**
- Date and Time data types store a date or a date/time value.

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

Database Structure



SQL Create Database

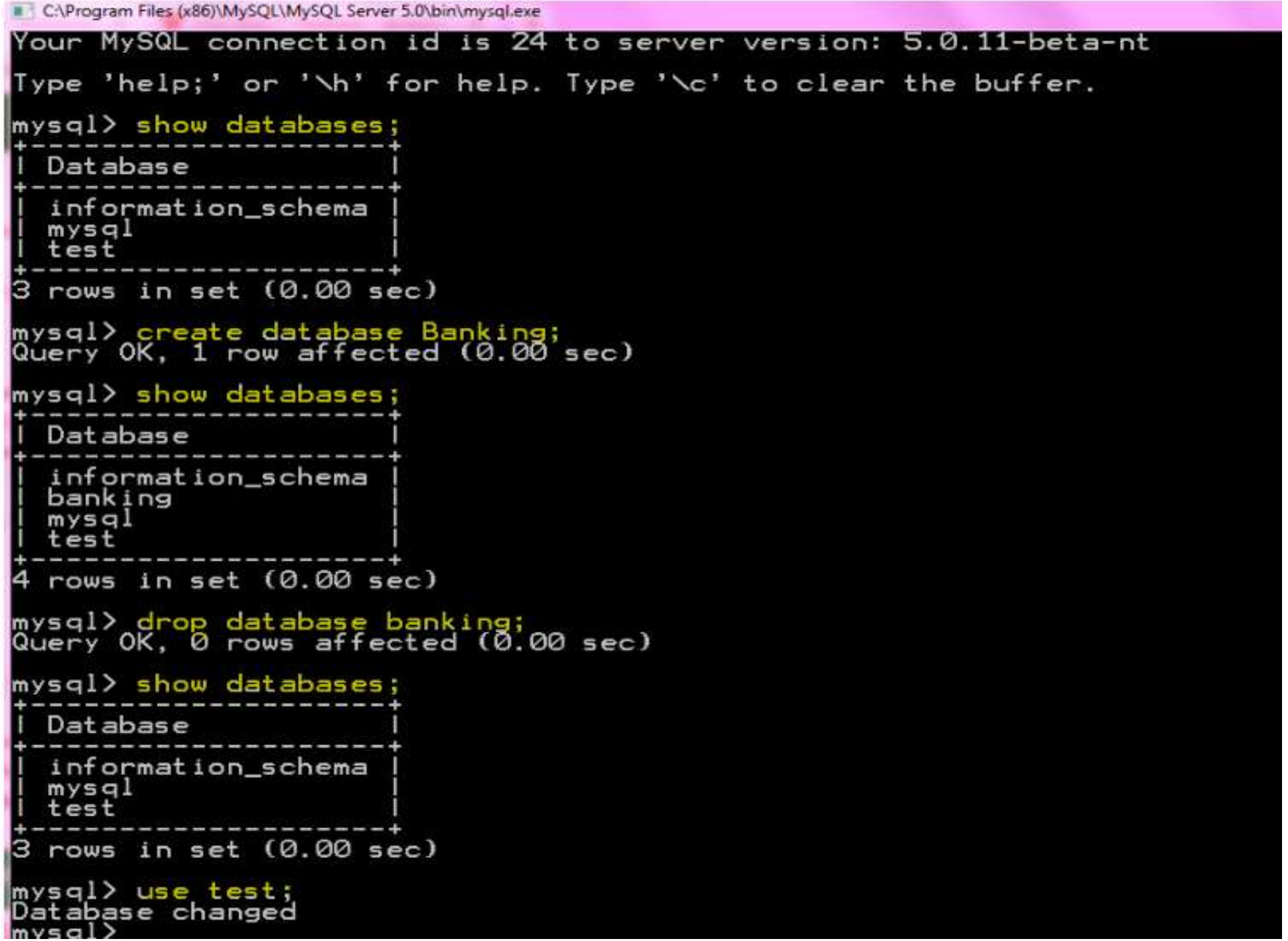
- In SQL, the 'Create Database' statement is a first step for storing the structured data in the database.
- Syntax of Create Database statement in SQL
- **CREATE DATABASE Database_Name;**
- In this syntax, **Database_Name** specifies the name of the database which we want to create in the system. We have to type the database name in query just after the 'Create Database' keyword.
- The database we want to create should be a simple and unique name, which can be easily identified.
- Database name should be no more than 128 characters.
- **Example**
- This example creates the **Student** database. To create the Student database, you have to type the following command in Structured Query Language:
- CREATE DATABASE Student ;

SQL DROP Database

- The SQL Drop Database statement deletes the existing database permanently from the database system. This statement deletes all the views and tables if stored in the database, so be careful while using this query in SQL.
- Syntax of Drop Database Statement in SQL
- **DROP DATABASE Database_Name;**
- **Example**
- DROP DATABASE Student;

SQL SELECT Database

- Suppose database users and administrators want to perform some operations on tables, views, and indexes on the specific existing database in SQL. Firstly, they have to select the database on which they want to run the database queries.
- Any database user and administrator can easily select the particular database from the current database server using the **USE** statement in SQL.
- Syntax of USE statement in SQL
- **USE database_name;**
- **Example**
- USE Student;
- Syntax of Show databases in SQL
- Suppose, we want to delete the Student database with all its data from the database system so, firstly we have to check that the Student database exists in the system or not by using the following statement:
- **SHOW DATABASES ;**
- **Display all the tables present in the database**
- **Syntax**
- **SHOW TABLES;**


C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
Your MySQL connection id is 24 to server version: 5.0.11-beta-nt
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
| mysql |
| test |
+--------------------+
3 rows in set (0.00 sec)
mysql> create database Banking;
Query OK, 1 row affected (0.00 sec)
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
| banking |
| mysql |
| test |
+--------------------+
4 rows in set (0.00 sec)
mysql> drop database banking;
Query OK, 0 rows affected (0.00 sec)
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
| mysql |
| test |
+--------------------+
3 rows in set (0.00 sec)
mysql> use test;
Database changed
mysql>

SQL Comment

- Comments can make your application easier for you to read and maintain. For example, you can include a comment in a statement that describes the purpose of the statement within your application.
- There are three syntaxes that you can use to create a comment within your SQL statement in MySQL.
- Syntax using # symbol.
- Begin the comment with -- (two hyphens).
- Begin the comment with a slash and an asterisk (/*). Proceed with the text of the comment. This text can span multiple lines. End the comment with an asterisk and a slash (*/).

```
1  -- Single Line Comment
2  # Single Line Comment
3  /*
4   a
5   b   Multi Line comment
6   c
7   */
```

The SQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a new table in a database.
- Syntax
- **CREATE TABLE table_name (**

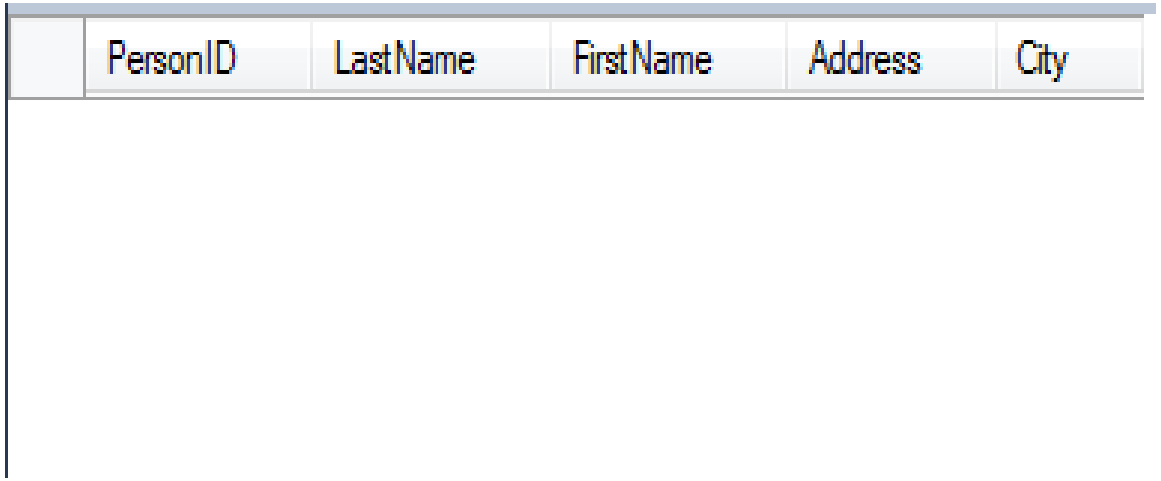
  column1 datatype,

  column2 datatype,

  column3 datatype,

  ....);
- The column parameters specify the names of the columns of the table.
- The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

- **Example**
- CREATE TABLE Persons (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
);



The SQL DROP TABLE Statement

- The DROP TABLE statement is used to drop an existing table in a database.
- Syntax
- **DROP TABLE table_name;**
- Example
- DROP TABLE Persons;

SQL ALTER TABLE Statement

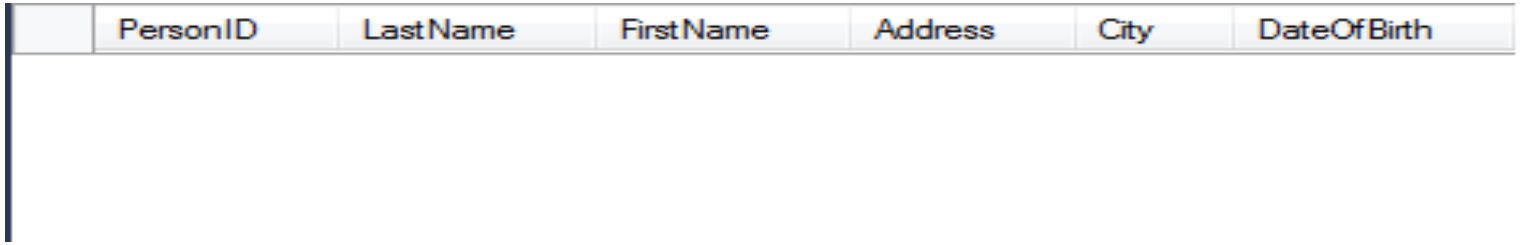
- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- **ALTER TABLE - ADD Column in last position by default**

To add a column in a table, use the following syntax:

ALTER TABLE table_name ADD column_name datatype;

- **Example**

ALTER TABLE Persons ADD DateOfBirth date;



- **ALTER TABLE - ADD Column in First Position**

To add a column in a table, use the following syntax:

ALTER TABLE table_name ADD column_name datatype first;

- **Example**

  ALTER TABLE Persons ADD DateOfBirth date first;

- **ALTER TABLE - ADD Column in Middle Position**

To add a column in a table, use the following syntax:

ALTER TABLE table_name ADD column_name datatype after column_name;

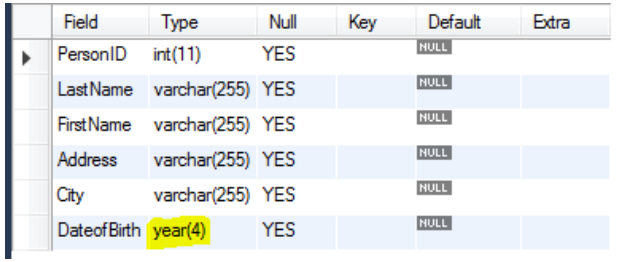
- **Example**

  ALTER TABLE Persons ADD DateOfBirth date after PersonID;

- **ALTER TABLE - MODIFY COLUMN**
- To change the data type of a column in a table, use the following syntax:
- **ALTER TABLE table_name MODIFY COLUMN column_name datatype;**

Example

Alter table persons modify column DateofBirth year;

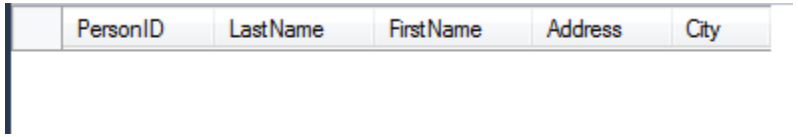


Field	Type	Null	Key	Default	Extra
PersonID	int(11)	YES		NULL	
LastName	varchar(255)	YES		NULL	
FirstName	varchar(255)	YES		NULL	
Address	varchar(255)	YES		NULL	
City	varchar(255)	YES		NULL	
DateofBirth	year(4)	YES		NULL	

- **ALTER TABLE - DROP COLUMN**
- To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):
- **ALTER TABLE table_name DROP COLUMN column_name;**

Example

ALTER TABLE Persons DROP COLUMN DateOfBirth;



PersonID	LastName	FirstName	Address	City

- **ALTER TABLE - RENAME COLUMN**
- **You can rename column use the following syntax:**
- **Alter Table Table_Name Rename column old_name to new_name**
- **Example**
- Alter Table persons Rename column DateOfBirth to Birth_Date;

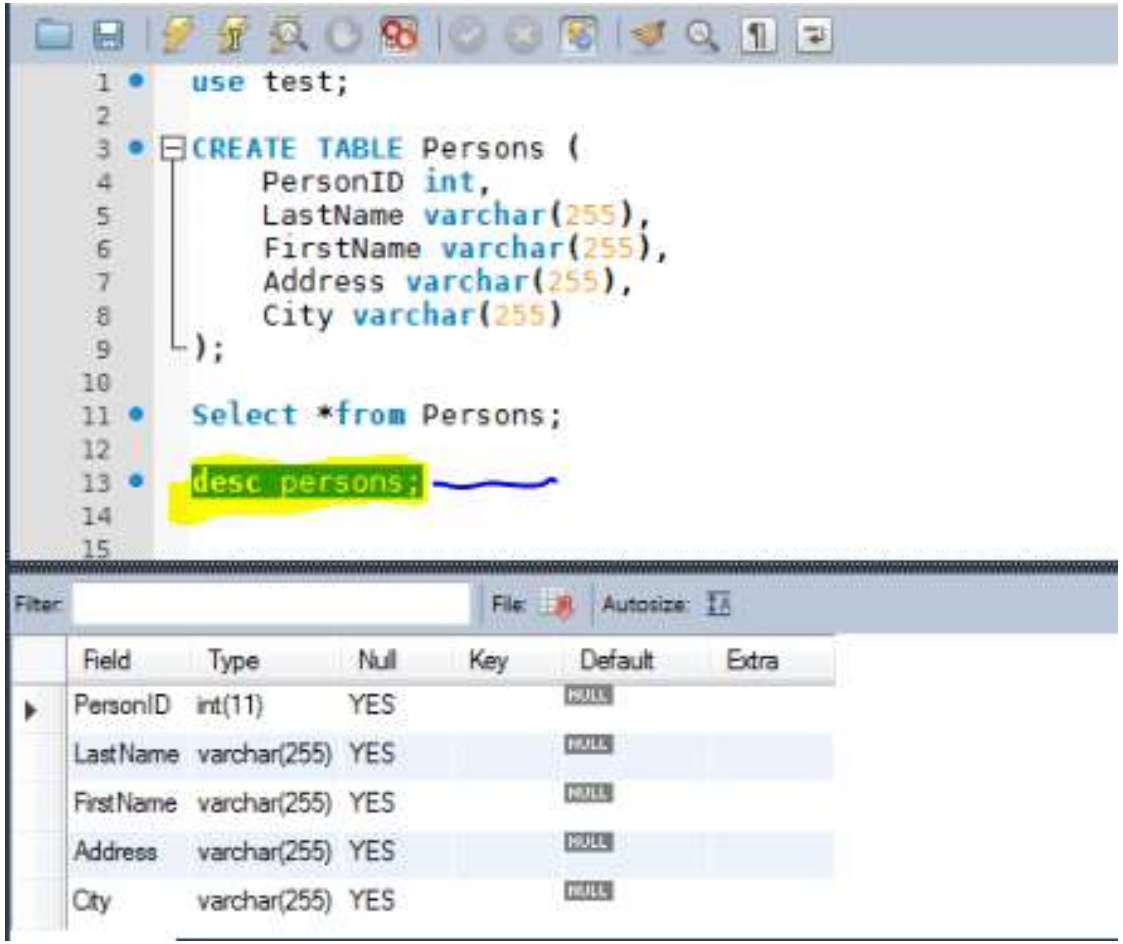
SQL TRUNCATE TABLE

- A truncate SQL statement is used to remove all rows (complete data) from a table.
- **TRUNCATE TABLE table_name;**
- Eg.
- TRUNCATE TABLE Persons;

•SQL RENAME TABLE

- **Syntax**
- **RENAME TABLE table_name1 to new_table_name1;**
- Eg.
- Rename table Employee to EMP;

- **Desc Table :** The table structure can be described by using Describe command.
- Syntax
- **Desc table_name;**
- Example
- Desc persons;



Tables in SQL

Employee Table

Emp_ID	Emp_Name	Salary
2	Amol	20500
1	Arpit	3000
3	Sita	4000
4	Gita	5000
5	Rahul	6000
6	Amol	14000
67	Rohan	NULL
69	Mamta	NULL
50	Rohan	19000

DML-SQL INSERT Query

- INSERT- INSERT statement is used to insert a single record or multiple records into a table in SQL Server.
- Syntax
- **INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]**

  VALUES (value1, value2, value3,...valueN);
- Here, column1, column2,...columnN are the names of the columns in the table into which you want to insert data.
- You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. The SQL INSERT
- INTO syntax would be as follows:
- **INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);**

- Create table Employee(Emp_ID int, Emp_Name varchar(255), Salary int);
- insert into Employee values(2, 'Amol', 20500);
- insert into Employee values(1,'Arpit', 3000);
- insert into Employee values(3,'Sita', 4000);
- insert into Employee values(4, 'Gita', 5000);
- insert into Employee values(5,'Rahul', 6000);
- insert into Employee values(6, 'Amol', 14000);
- insert into Employee(Emp_ID,Emp_Name)values(67,'Rohan');
- insert into Employee(Emp_ID,Emp_Name)values(69,'Mamta');
- insert into Employee values(50, 'Rohan', 19000);

SQL WHERE Clause

- The SQL WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables.
- If the given condition is satisfied, then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.
- The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement.
- **Syntax:**
- **SELECT column1, column2, columnN FROM table_name WHERE [condition]**
- **Example**
- Select Emp_Name from Employee where Emp_ID=1
- Select *from Employee where Emp_Name='Amol'

DML-SQL UPDATE Query

- The SQL UPDATE Query is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update selected rows, otherwise all the rows would be affected.
- Syntax
- **UPDATE table_name SET column1 = value1, column2 = value2...., columnN = valueN WHERE [condition];**
- **Example**
- Update Employee set Emp_Name='Sunita' where Emp_ID=3;

Emp_ID	Emp_Name	Salary
2	Amol	20500
1	Arpit	3000
3	Sunita	4000
4	Gita	5000
5	Rahul	6000
6	Amol	14000
67	Rohan	NULL
69	Mamta	NULL
50	Rohan	19000

DML-SQL DELETE Query

- The SQL DELETE Query is used to delete the existing records from a table. You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.
- Syntax
- **DELETE FROM table_name WHERE [condition];**
- **Example**
- Delete from Employee where Emp_ID=5;
- If you want to DELETE all the records from Employee table, you do not need to use WHERE clause and DELETE query would be as follows:
- Delete from Employee;

DQL-SQL SELECT Query

- SQL SELECT Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Emp_ID	Emp_Name
2	Amol
1	Arpit
3	Sita
4	Gita
5	Rahul
6	Amol
67	Rohan
69	Mamta
50	Rohan

- Syntax:
- **SELECT column1, column2, columnN FROM table_name;**
- **Example**
- Select Emp_ID, Emp_Name from Employee;
- Here, column1, column2...are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax:
- **SELECT * FROM table_name;**
- **Example**
- Select *from Employee

Emp_ID	Emp_Name	Salary
2	Amol	20500
1	Arpit	3000
3	Sita	4000
4	Gita	5000
5	Rahul	6000
6	Amol	14000
67	Rohan	NULL
69	Mamta	NULL
50	Rohan	19000

MySQL ENUM Data type example

- The ENUM data type in MySQL is a string object. It allows us to limit the value chosen from a list of permitted values in the column specification at the time of table creation. It is short for enumeration, which means that each column may have one of the specified possible values. It uses numeric indexes (1, 2, 3…) to represent string values.
- CREATE TABLE shirts (id INT, name VARCHAR(35), size ENUM('small', 'medium', 'large', 'x-large'));
- INSERT INTO shirts(id, name, size) VALUES (1,'t-shirt', 'medium'), (2, 'casual-shirt', 3), (3, 'formal-shirt', 4), (4, 'polo-shirt', 'small');
- INSERT INTO shirts values(1,'mnmn','**mm**'); **-- getting error because insering different value which not specified in ENUM**


There are 3 types of Transaction Control Statements:
1 Commit
2 Rollback
3 Savepoint

TCL - Commit

- The COMMIT command in SQL is used to **permanently save any transaction into the database**..
- Syntax

```
COMMIT;
```

TCL - Rollback

- The Rollback command undo any changes made to the database
- **For example**, suppose we have used the UPDATE command at any point to make certain changes to our database and later realize that those changes need to be reverted(or undone), in that case, we can use the ROLLBACK command. The rollback command will basically revert or roll back any changes that were not committed during our transaction using the COMMIT command.
- Syntax

```
ROLLBACK;
```

TCL - Savepoint

- The SAVEPOINT command in TCL is basically used to temporarily save a transaction so that we can roll back to that point (saved point) whenever required.
- Syntax

```
SAVEPOINT savepoint_name;
```

Interestingly, the rollback command can also be used with the **SAVEPOINT** command to jump to a savepoint in any ongoing transaction. The savepoints are like checkpoints, they temporarily save a transaction up to where the transaction can be rolled back

- Syntax

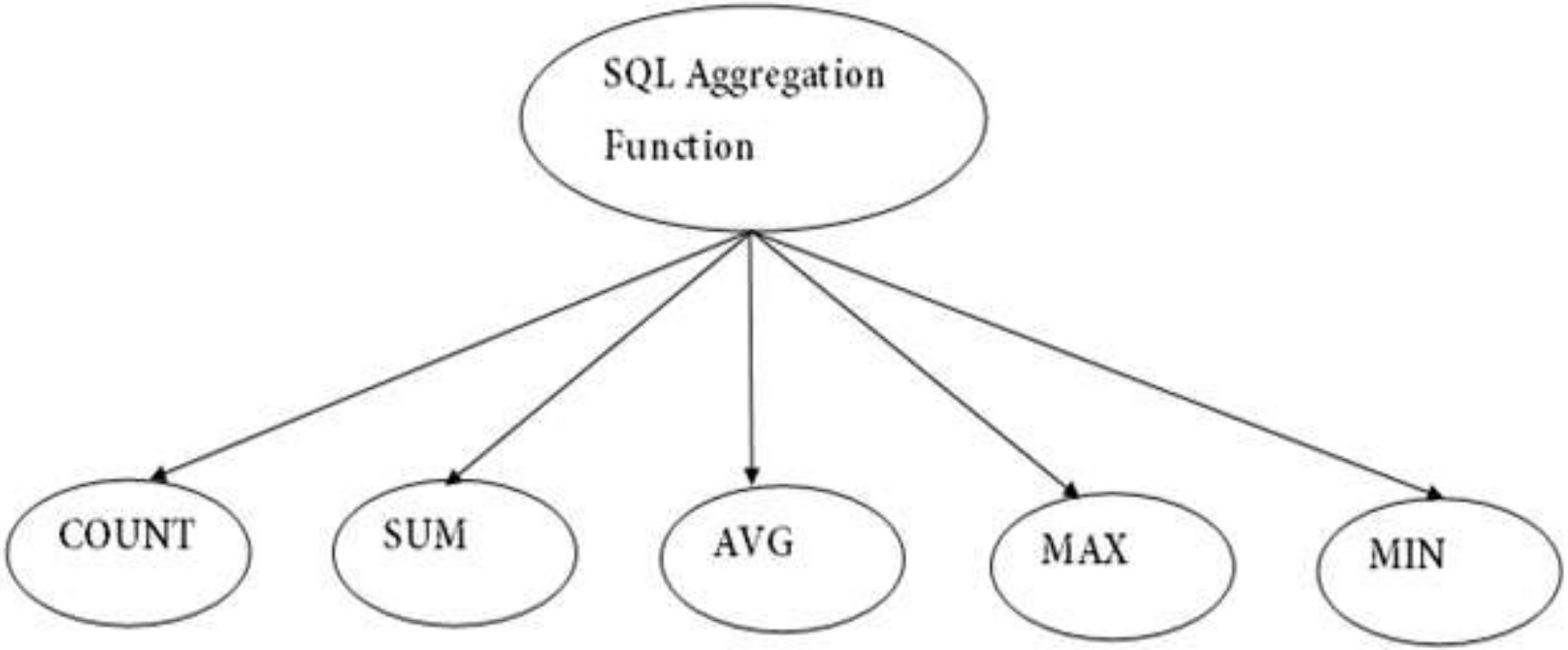
```
ROLLBACK TO savepoint_name;
```

DDL	DML
It stand for Data Definition Language	It stand for Data Manipulation Language
Basic command present in DDL are – Create, Drop, Alter, Truncate, Rename	Basic command present in DML are – Insert, Update, Delete
DDL does not use Where clause in its statement.	While DML uses WHERE clause in its statement.
DDL is used to define the structure of a database.	DML is used to manipulate the data within the database.
DDL statements do not change the contents of the database.	DML statements change the contents of the database.

	DELETE Command	DROP Command	TRUNCATE Command
Language	The DELETE command is Data Manipulation Language Command.	The DROP command is Data Definition Language Command.	The TRUNCATE command is a Data Definition Language Command.
Use	The DELETE command deletes one or more existing records from the table in the database.	The DROP Command drops the complete table from the database.	The TRUNCATE Command deletes all the rows from the existing table, leaving the row with the column names.
Transition	We can restore any deleted row or multiple rows from the database using the ROLLBACK command.	We cannot get the complete table deleted from the database using the ROLLBACK command.	We cannot restore all the deleted rows from the database using the ROLLBACK command.
Syntax	DELETE FROM table_name WHERE condition;	DROP TABLE table_name;	TRUNCATE TABLE table_name;

Aggregate Function

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.



- **COUNT()** function returns the number of rows that matches.
- **AVG()** function returns the average value of a numeric column.
- **SUM()** function returns the total sum of a numeric column.
- **MAX()** function returns the max number of rows that matches a specified criterion.
- **MIN()** function returns the min value of a numeric column.
- **Syntax**
- **SELECT COUNT (column_name) FROM table_name;**
- **SELECT AVG (column_name) FROM table_name;**
- **SELECT SUM (column_name) FROM table_name;**
- **SELECT MAX (column_name) FROM table_name;**
- **SELECT MIN(column_name) FROM table_name;**

- **Example**
- Select Count(Salary) from Employee;
- Select SUM(Salary) from Employee;
- Select AVG(Salary) from Employee;
- Select MAX(Salary) from Employee;
- Select MIN(Salary) from Employee;

Count(Salary)
7

SUM(Salary)
71500

AVG(Salary)
10214.2857

MAX(Salary)
20500

MIN(Salary)
3000

- Employee Table

Emp_ID	Emp_Name	Salary
2	Amol	20500
1	Arpit	3000
3	Sita	4000
4	Gita	5000
5	Rahul	6000
6	Amol	14000
67	Rohan	NULL
69	Mamta	NULL
50	Rohan	19000

- Select MAX(Salary) from Employee where Emp_ID > 5;

MAX(Salary)
19000

- Select MIN(Salary) from Employee where Emp_ID > 4;

MIN(Salary)
6000

- Select SUM(Salary) from Employee where Salary < 10000;

SUM(Salary)
18000

- Select AVG(Salary) from Employee where Emp_ID < 4;

AVG(Salary)
9166.6667

- Select count(*)from Employee;

count(*)
9

- Select count(*)from Employee where Emp_ID < 10;

count(*)
6

String Function

1. **ASCII** - Returns the ASCII value for the specific character

Syntax

Select ASCII(character);

Example

Select ASCII('S');

2. **Concat** – This function is used to add two words or Strings tighter

Syntax

Select column_name, concat(String1, String2,…) from table_name;

Example

• Select department, concat(first_name, ' has id –', emp_no) as Employees_code from employees;

String Function

3. **Concat_ws** – This function is used to concatenate multiple strings together with a specified separator.

Syntax

Select column_name, concat_ws(separator, string1,string2,..) from table_name;

Example

Select department, concat_ws('_', emp_no, first_name, department) as concat_string from employees;

4. **Length** – This function is used to find the Length of a word or string

Syntax

Select column_name, length(column_name) from table_name;

Example

Select emp_no, first_name, length(First_name) as name_length from employees;

String Function

5. **Upper**– This function is used to make the string in uppercase

Syntax

Select column_name, upper(column_name) from table_name;

Example

select emp_no, upper(first_name) as result from employees;

6. **Lower** – This function is used to make the string in lowercase

Syntax

Select column_name, lower(column_name) from table_name;

Example

select emp_no, lower(first_name) as result from employees;

String Function

7. **Replace**– Replace all the occurrences of a substring withing a string

Syntax

Select column_name, replace(string_expression, search_string, replacement_string)from table_name;

Example

Select emp_no, first_name, replace(department,'HR','Human Resource') as result from employees;

8. **Reverse** – This function is used to reverse a string

Syntax

Select column_name, reverse(column_name) from table_name;

Example

select emp_no, reverse(first_name) as result from employees;

String Function

9. **Substring**– It allows you to extract a portion(substring) of a string based on a specified starting position and length.

Syntax

Select column_name, substring(string_expression, start_position, length)from table_name;

string_expression – The string from which you want to extract a substring

Start_position – The starting position of a substring within the string. It is usually 1-based. (the first character is at position 1)

Example

Select emp_no,first_name,substring(birth_date,1,4)as result from employees;

String Function

10. **Left**– Returns a specified number of character from the left side of a string

Syntax

Select column_name, left(String_expression, length) from table_name;

Example

Select emp_no, left(first_name,2) as result from employees;

11. **Right** – Returns a specified number of character from the Right side of a string

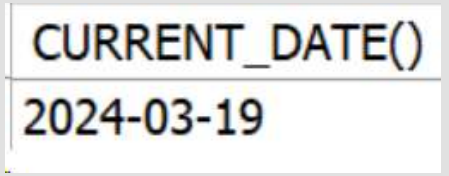
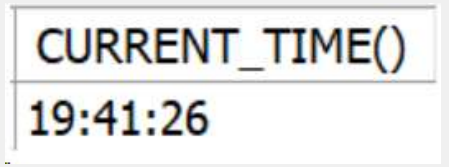
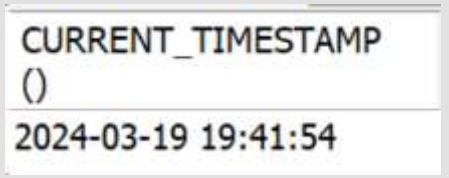
Syntax

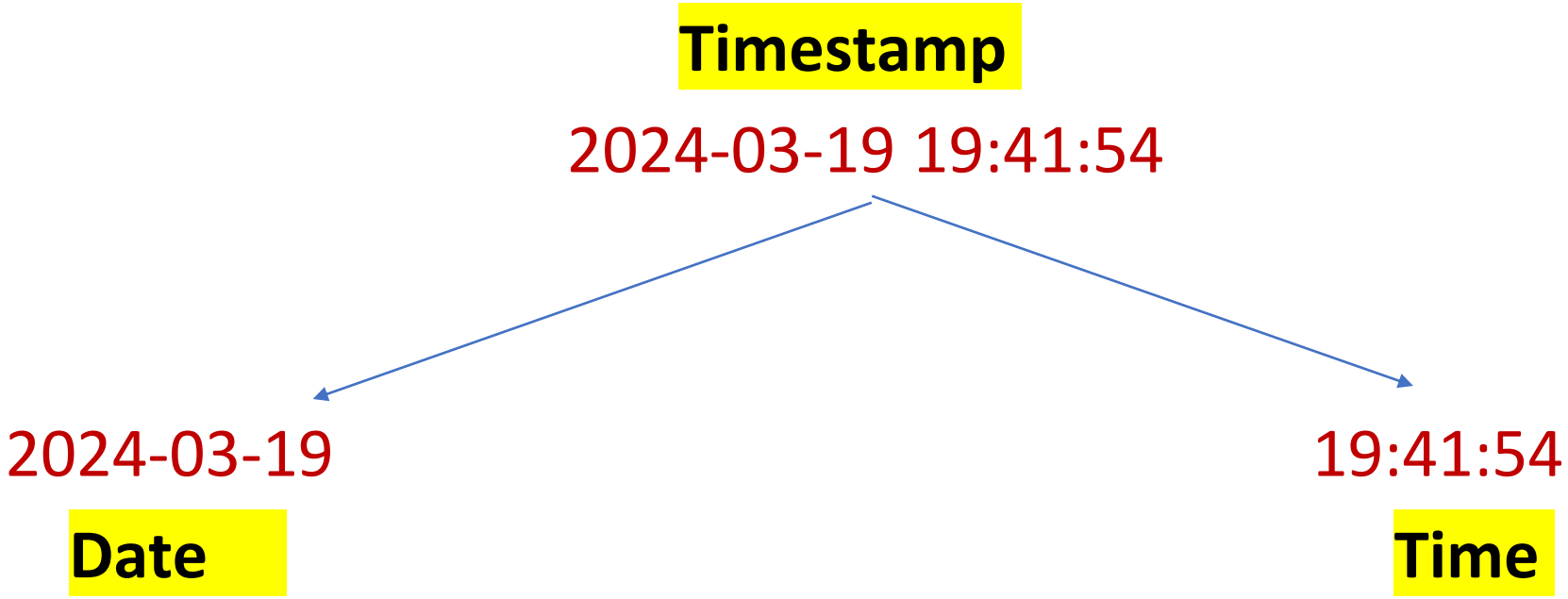
Select column_name, right(String_expression, length) from table_name;

Example

Select emp_no, right(first_name,2) as result from employees;

Date and Time Function

Date and Time Function	Output
Select CURRENT_DATE(); -- Return the current date	
Select CURRENT_TIME (); -- Return the current Time	
Select CURRENT_TIMESTAMP (); -- Return current Timestamp	



- **Date – Extract the date part from a datetime expression (timestamp)**
- **Time – Extract the time part from a datetime expression(timestamp)**

```
select date('2024-03-19 19:54:35’);
select Time('2024-03-19 19:54:35’);
```

- **Year** – Extract the year from date or datetime expression
- **Month** – Extract the month from date or datetime expression
- **Day** - Extract the Day from date or datetime expression
- **Hour** – Extract the Hour from date or datetime expression
- **Minute** – Extract the Minute from date or datetime expression
- **Second** – Extract the Second from date or datetime expression

```
select Month('2024-03-19 19:54:35’);
select Day('2024-03-19 19:54:35’);
select Year('2024-03-19 19:54:35’);
select Hour('2024-03-19 19:54:35’);
select Minute('2024-03-19 19:54:35’);
select Second('2024-03-19 19:54:35');
```

Extract – Extract a specific component (Year, Month, day etc.)

From a date, time or timestamp

Syntax

Select Extract(component from datetime_expression);

Example

Q – Return the emp_no, first_name and only the day from the date of birth from the employee table

Select emp_no, first_name, extract(day from birth_date)from employees;

DateDiff – Calculate the difference between two dates and or Timestamp

Select datediff(date1, date2); --date1 > date2

Select datediff('2024-02-01','2024-1-1');

TimeDiff – Calculate the difference between two times or timestamps

```
Select TimeDiff(time1,time2);
```

```
Select timediff('10:30:00','10:15:00');
```

SQL Distinct Keyword

- The SQL DISTINCT keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.
- There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.
- **Syntax**
- **SELECT DISTINCT column_name FROM table_name**
- **Example**
- Select distinct Emp_Name from Employee;

SQL Operators

- A n operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.
- Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.
- **Arithmetic operators**
- **Comparison operators**
- **Logical operators**
- **Operators used to negate conditions**

SQL Arithmetic Operators:

- Assume **'variable a'** holds 10 and **'variable b'** holds 20, then –

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

```
mysql> select 10+20;
+-------+
| 10+20 |
+-------+
|    30 |
+-------+
1 row in set (0.00 sec)

mysql> select 10*20;
+-------+
| 10*20 |
+-------+
|   200 |
+-------+
1 row in set (0.00 sec)

mysql> select 10/5;
+--------+
| 10/5   |
+--------+
| 2.0000 |
+--------+
1 row in set (0.00 sec)

mysql> select 12 % 5;
+--------+
| 12 % 5 |
+--------+
|      2 |
+--------+
1 row in set (0.00 sec)
```

Que: How to increase 10% salary of the employee

```
UPDATE Employee
  SET Salary = Salary + (Salary * 10/100);
```

SQL Comparison Operators:

- Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

- **Example**
- Select *from Employee where Salary > 5000
- Select *from Employee where Salary < 5000
- Select *from Employee where Salary = 20500
- Select *from Employee where Salary != 20500
- Select *from Employee where Salary >= 5000
- Select *from Employee where Salary <= 5000
- Select *from Employee where Salary <> 5000

SQL Logical Operators:

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL AND Operator

- The **AND operator** in SQL would show the record from the database table if all the conditions separated by the AND operator evaluated to True. It is also known as the conjunctive operator and is used with the WHERE clause.
- Syntax
- **SELECT column1, column2, ...**
  FROM table_name
  WHERE condition1 AND condition2 AND condition3 ...;
- **Example**
- Select Emp_Name from Employee where Emp_ID=1 AND Emp_Name='Arpit';
- Select *from Employee where Emp_ID < 10 AND Salary > 10000;

SQL OR Operator

- The OR operator in SQL shows the record from the table if any of the conditions separated by the OR operator evaluates to True. It is also known as the conjunctive operator and is used with the WHERE clause.
- **Syntax**
- **SELECT column1, column2, ...**
  FROM table_name
  WHERE condition1 OR condition2 OR condition3 ...;
- **Example**
- Select Emp_Name from Employee where Emp_ID=90 OR Emp_Name='Arpit';
- Select *from Employee where Salary > 6000 OR Salary < 1000;

SQL NOT Operator

- The NOT operator displays a record if the condition(s) is NOT TRUE.
- **Syntax**
- **SELECT column1, column2, ...**
  FROM table_name
  WHERE NOT condition;
- **Example**
- Select *from Employee where Not Emp_ID=2;
- Select *from Employee where Not Emp_Name='Rohan';

SQL Between Operator

- The **BETWEEN operator** in SQL shows the record within the range mentioned in the SQL query.
- **Syntax**
- **SELECT column_name FROM table_name WHERE column_name BETWEEN Value1 AND Value2**
- **Example**
- Select *from Employee where Salary between 2000 AND 6000

SQL IN Operator

- The **IN operator** in SQL allows database users to specify two or more values in a WHERE clause. This logical operator minimizes the requirement of multiple OR conditions.
- **Syntax**
- **SELECT column_name FROM table_name WHERE column_name IN (Value1, Value2, Value3)**
- **Example**
- Select *from Employee where Salary IN(4000,6000,20500)

IS NULL & IS NOT NULL

- keyword / operator used with WHERE clause.
- To test NULL values, We will have to use the IS NULL and IS NOT NULL operators
- **Is Null Syntax**
- **SELECT column_names FROM table_name WHERE column_name IS NULL;**
- **Example**
- Select *from Employee where Salary IS NULL
- **Is Not Null Syntax**
- **SELECT column_names FROM table_name WHERE column_name IS NOT NULL;**
- **Example**
- Select *from Employee where Salary IS NOT NULL

The SQL LIKE Operator

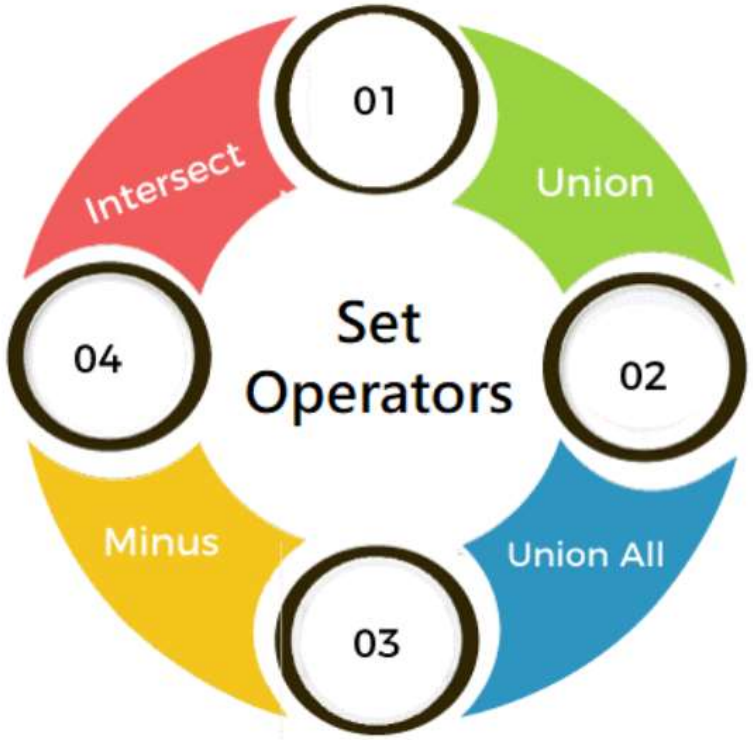
- The Like Operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character
- LIKE Syntax
- **SELECT column1, column2, ...**
  FROM table_name
  WHERE columnN LIKE pattern;

Here are some examples showing different `LIKE` operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Set Operators

- SET operators are special type of operators which are used to *combine the result of two queries.*
- Operators covered under SET operators are:

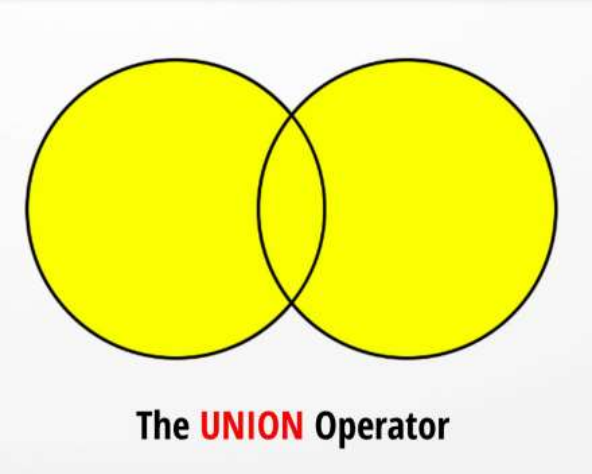


Union

- UNION clause/operator is used to combine the result sets of 2 or more SELECT statements. It removes duplicate rows between the various SELECT statements.
- Each SELECT statement within the UNION operator must have the same number of columns in the result sets with similar data types.
- **Syntax**
- **SELECT column_name FROM table1**

  UNION

  SELECT column_name FROM table2;
- **Example**
- SELECT * FROM First UNION SELECT * FROM Second;

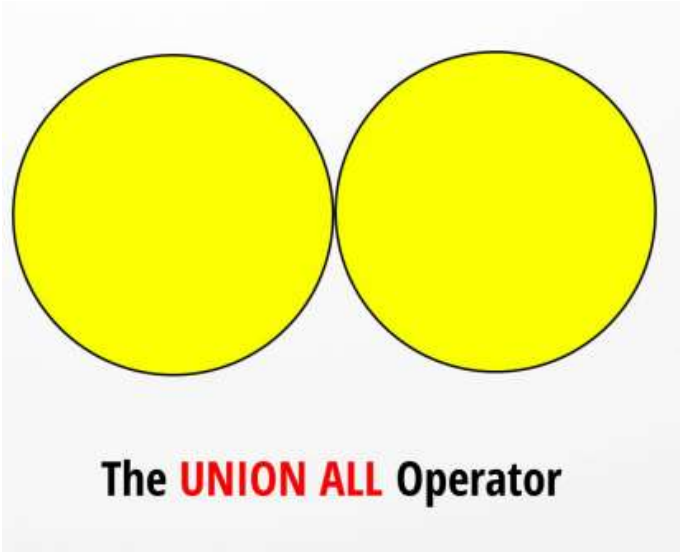


Union All

- The UNION ALL clause/operator is used to combine the results of two SELECT statements including duplicate rows.
- **Syntax**
- **SELECT column_name FROM table1**

  UNION ALL

  SELECT column_name FROM table2;
- Example
- SELECT * FROM First UNION ALL SELECT * FROM Second;

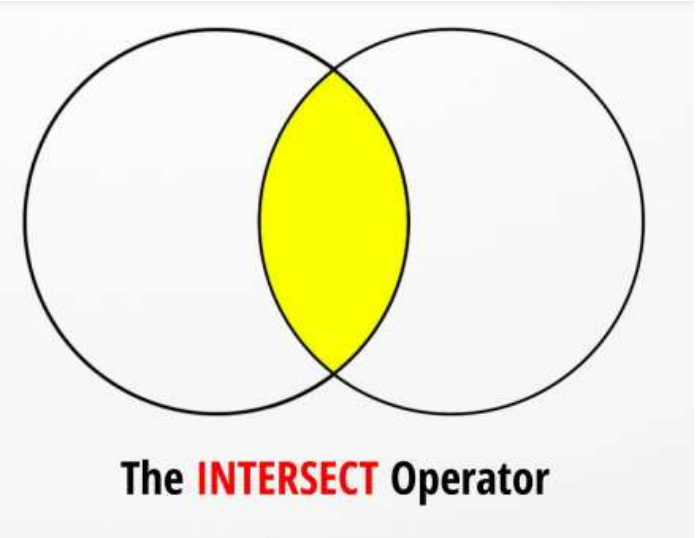


Intersect

- **The INTERSECT operator returns the distinct (common) elements in two sets or common records from two or more tables.**
- **Syntax**
- **SELECT column_name FROM table1**

  Intersect

  SELECT column_name FROM table2;



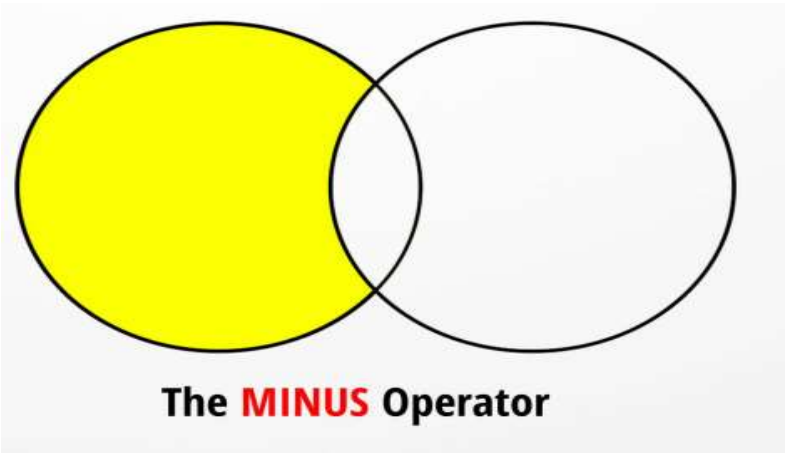
- Example
- SELECT * FROM First Intersect SELECT * FROM Second;

Minus

- **The MINUS operator returns the unique element from the first table/set, which is not found in the second table/set**
- **Syntax**
- **SELECT column_name FROM table1**

  Minus

  SELECT column_name FROM table2;



- Example
- SELECT * FROM First Minus SELECT * FROM Second;

```
mysql> Select *from First;
+------+---------+
| ID   | Name    |
+------+---------+
|    1 | Jack    |
|    2 | Harry   |
|    3 | Jackson |
+------+---------+
3 rows in set (0.00 sec)

mysql> Select *from Second;
+------+---------+
| ID   | Name    |
+------+---------+
|    3 | Jackson |
|    4 | Stephan |
|    5 | David   |
+------+---------+
3 rows in set (0.00 sec)

mysql> Select *from First Union Select *from Second;
+------+---------+
| ID   | Name    |
+------+---------+
|    1 | Jack    |
|    2 | Harry   |
|    3 | Jackson |
|    4 | Stephan |
|    5 | David   |
+------+---------+
```

```
mysql> Select *from First Union All Select *from Second;
+------+---------+
| ID   | Name    |
+------+---------+
|    1 | Jack    |
|    2 | Harry   |
|    3 | Jackson |
|    3 | Jackson |
|    4 | Stephan |
|    5 | David   |
+------+---------+
6 rows in set (0.00 sec)

mysql> Select *from First Intersect Select *from Second;
+------+---------+
| ID   | Name    |
+------+---------+
|    3 | Jackson |
+------+---------+
1 row in set (0.00 sec)

mysql> Select *from First Except Select *from Second;
+------+-------+
| ID   | Name  |
+------+-------+
|    1 | Jack  |
|    2 | Harry |
+------+-------+
2 rows in set (0.00 sec)
```

MySQL Case Statement

- The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.
- If there is no ELSE part and no conditions are true, it returns NULL.
- **Syntax**

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

MySQL Case Statement

- **Example**

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30’
    WHEN Quantity = 30 THEN 'The quantity is 30’
    ELSE 'The quantity is under 30’
    END AS QuantityText
FROM OrderDetails;
```

SQL Statement:

```
SELECT OrderID, Quantity,
CASE WHEN Quantity > 30 THEN 'The quantity is greater than 30'
WHEN Quantity = 30 THEN 'The quantity is 30'
ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

Edit the SQL Statement, and click "Run SQL" to see the result.

Run SQL »

Result:

Number of Records: 2155

OrderID	Quantity	QuantityText
10248	12	The quantity is under 30
10248	10	The quantity is under 30
10248	5	The quantity is under 30
10249	9	The quantity is under 30
10249	40	The quantity is greater than 30

IQ – How to delete duplicate rows in Table

Create table sample(id int, s_name varchar(255));
Insert into sample values(1,'aa');
Insert into sample values(1,'aa');
Insert into sample values(1,'aa');

id	s_name
1	aa
1	aa
1	aa

Step -1 :

- Create an intermediate table that has the same structure as the source table and transfer the unique rows found in the source:

- CREATE TABLE copy_sample SELECT DISTINCT id, s_name from sample;

Step - 2:

With that done, you can delete the source table with the drop command and rename the new one:

- drop table sample;

- Alter table copy_sample rename to sample;

id	s_name
1	aa

IQ – SQL Query to fetch (print) Even and Odd records from a table

```
create table Emp(ID int, Name varchar(255), Contact int, Salary int);

Insert into emp
values(1,'Preeti',123456,2000),(2,'Arpit',123789,3000),(3,'Aaditya',147
258,5000),(4,'APOORV',789632,10000),(5,'RIDDHI',148796,20000),(6,'R
IYA',148965,30000),(7,'Wenkey',128659,40000),(8,'Sita',65656,7877);

select *from emp;
```

Case 1: Write a query to find even records in MYSQL.

```
SELECT * FROM EMP WHERE id IN(SELECT id FROM EMP WHERE id % 2 = 0);
```

Case 2: Write a query to find odd records in MYSQL

```
SELECT * FROM EMP WHERE id IN(SELECT id FROM EMP WHERE id % 2 != 0);
```

IQ – SQL Query to fetch (print) Even and Odd records from a table

- Another way

- **select * from EMP where mod(id, 2)=0;**

- **select * from EMP where mod(id, 2)=1;**

SQL Group By Clause

- The GROUP BY clause is a SQL command that is used to **group rows that have the same values**. The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.
- **Syntax**
- **Select Column, Sum(Column) from table GROUP BY column;**
- Example
- Select Company, SUM(Amount) from Sales Group By Company;

Example : Group By Clause

- Select Company, SUM(Amount) from Sales **Group By** Company;

Table: Sales

Company	Amount
Wipro	5500
IBM	4500
Wipro	7100
IBM	6700
HCL	8900
MindTree	12000

Output of Group By

Filter:

Company	SUM(Amount)
HCL	8900
IBM	11200
MindTree	12000
Wipro	12600

SQL Having Clause

- Having was added to SQL because the WHERE keyword could not be used against aggregate functions(like SUM), and without HAVING.. It would be impossible to test for result conditions.
- **Syntax**

```
SELECT column, SUM(Column) FROM table
GROUP BY column
HAVING SUM(column) condition value;
```

- **Example**
- Select Company, SUM(Amount) from Sales Group By Company having SUM(Amount)<10000;

Example: Having Clause

- Select Company, SUM(Amount) from Sales Group By Company having SUM(Amount)<10000;
- Table Sales

Company	Amount
Wipro	5500
IBM	4500
Wipro	7100
IBM	6700
HCL	8900
MindTree	12000

Output

Company	SUM(Amount)
HCL	8900

WHERE Clause	HAVING Clause
The WHERE clause does not allow to work with aggregate functions.	The HAVING clause can work with aggregate functions.
We can use the WHERE clause with the SELECT, UPDATE, and DELETE statements.	The HAVING clause can only use with the SELECT statement.

SQL Order By Clause

- The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.
- **Syntax**

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

- **Example**
- Select *from Sales Order By Amount ASC;
- Select *from Sales Order By Amount DESC;

Example: Order By Clause

- Table Sales

Company	Amount
Wipro	5500
IBM	4500
Wipro	7100
IBM	6700
HCL	8900
MindTree	12000

- **Select *from Sales Order By Amount ASC;**

Company	Amount
IBM	4500
Wipro	5500
IBM	6700
Wipro	7100
HCL	8900
MindTree	12000

- Table Sales

Company	Amount
Wipro	5500
IBM	4500
Wipro	7100
IBM	6700
HCL	8900
MindTree	12000

Select *from Sales Order By Amount DESC;

Company	Amount
MindTree	12000
HCL	8900
Wipro	7100
IBM	6700
Wipro	5500
IBM	4500

MySQL Limit Clause

The `LIMIT` clause is used in the `SELECT` statement to constrain the number of rows to return. The `LIMIT` clause accepts one or two arguments. The values of both arguments must be zero or positive integers.

The following illustrates the `LIMIT` clause syntax with two arguments:

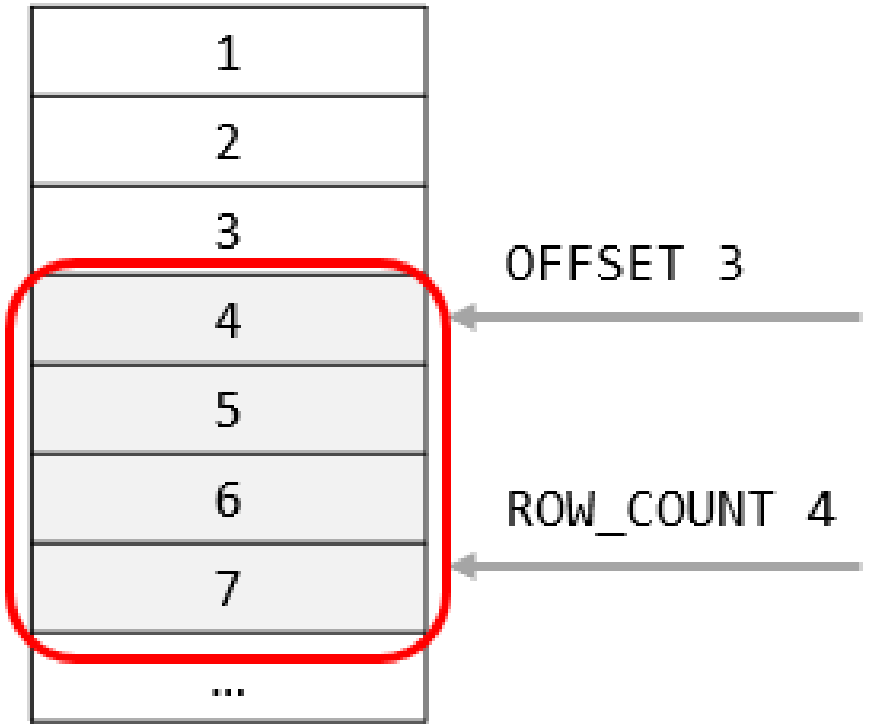
```
SELECT
    select_list
FROM
    table_name
LIMIT [offset,] row_count;
```

In this syntax:

- The `offset` specifies the offset of the first row to return. The `offset` of the first row is 0, not 1.
- The `row_count` specifies the maximum number of rows to return.

The following picture illustrates the LIMIT clause

```
SELECT n FROM t
ORDER BY n
LIMIT 3, 4;
```



Using MySQL LIMIT to get the n^{th} highest or lowest value

- To get the n^{th} highest or lowest value, you use the following Limit Clause

To get the n^{th} highest or lowest value, you use the following `LIMIT` clause:

```
SELECT select_list
FROM table_name
ORDER BY sort_expression
LIMIT n-1, 1;
```

The clause `LIMIT n-1, 1` returns `1` row starting at the row `n` .

Sales Table

Company	Amount
Wipro	5500
IBM	4500
Wipro	7100
IBM	6700
HCL	8900
MindTree	12000

Desc

Company	Amount
MindTree	12000
HCL	8900
Wipro	7100
IBM	6700
Wipro	5500
IBM	4500

- **Find the 3rd Highest Value**
- Select *from Sales Order by Amount Desc Limit 2, 1;

Company	Amount
Wipro	7100

- **Find the 2nd Highest Value**
- Select *from Sales Order by Amount Desc Limit 1, 1

Company	Amount
HCL	8900

- **Find the Highest Value in Table**
- Select *from Sales Order by Amount Desc Limit 0, 1

Company	Amount
MindTree	12000

- **Find the 4th Highest Value in Table**
- Select *from Sales Order by Amount Desc Limit 3, 1;

Company	Amount
IBM	6700

Find the 3rd Lowest Value

- Select *from Sales Order by Amount Asc Limit 2, 1;
- **Find the 2nd Lowest Value**
- Select *from Sales Order by Amount Asc Limit 1, 1;
- **Find the Lowest Value in Table**
- Select *from Sales Order by Amount Asc Limit 0, 1;
- **Find the 4th Lowest Value in Table**
- Select *from Sales Order by Amount Asc Limit 3, 1;

SQL Alias Syntax

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the As Keyword
- **Alias Column Syntax**
- **SELECT column_name AS alias_name**
  FROM table_name;
- **Alias Table Syntax**
- **SELECT column_name(s)**
  FROM table_name AS alias_name;

Example of Alias

- Consider the following two tables,

- (a) CUSTOMERS table

ID	Name	Age	Address	Salary
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

(b) Orders Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08	3	3000
100	2009-10-08	3	1500
101	2009-11-20	2	1560
103	2008-05-20	4	2060

- Now, following is the usage of **table alias:**
- SELECT C.ID, C.NAME, C.AGE, O.AMOUNT FROM CUSTOMERS AS **C**, ORDERS AS **O** WHERE C.ID = O.CUSTOMER_ID;

ID	NAME	AGE	AMOUNT
2	Khilan	25	1560
3	kaushik	23	3000
3	kaushik	23	1500
4	Chaitali	25	2060

Example of Alias

CUSTOMERS table

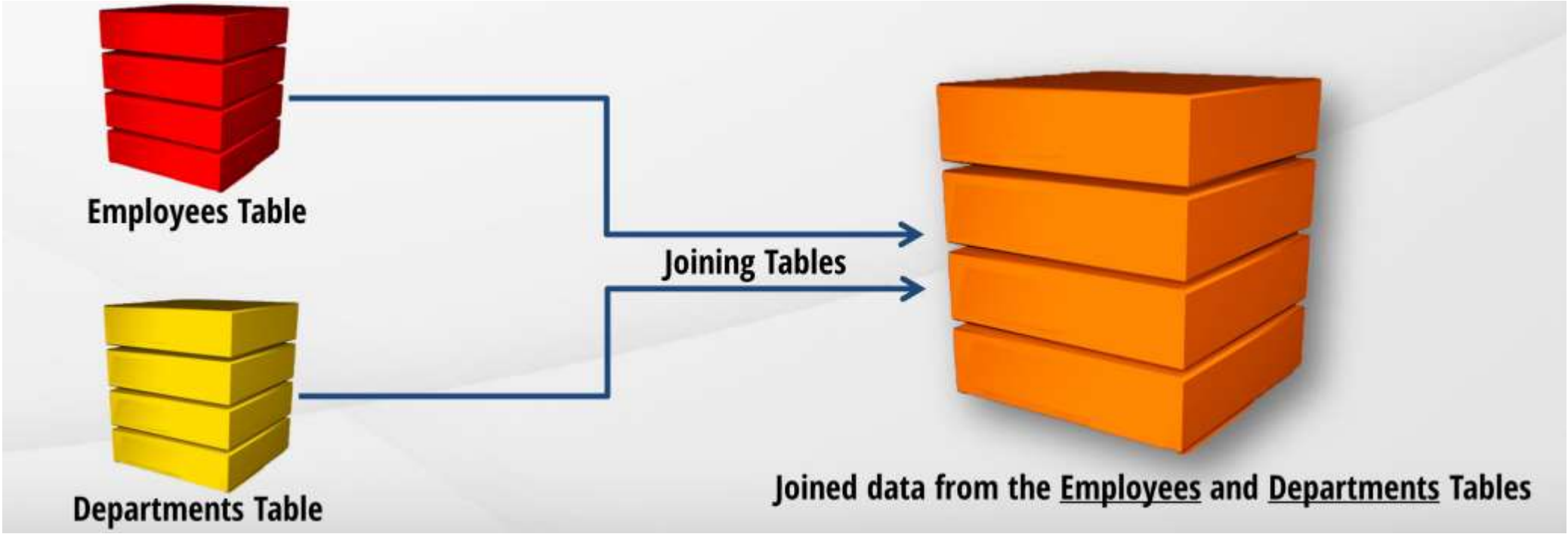
ID	Name	Age	Address	Salary
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

- Now, following is the usage of **Column alias:**
- SELECT ID AS CUSTOMER_ID, NAME AS CUSTOMER_NAME FROM CUSTOMERS;

CUSTOMER_ID	CUSTOMER_NAME
1	Ramesh
2	Khilan
3	kaushik
4	Chaitali
5	Hardik
6	Komal
7	Muffy

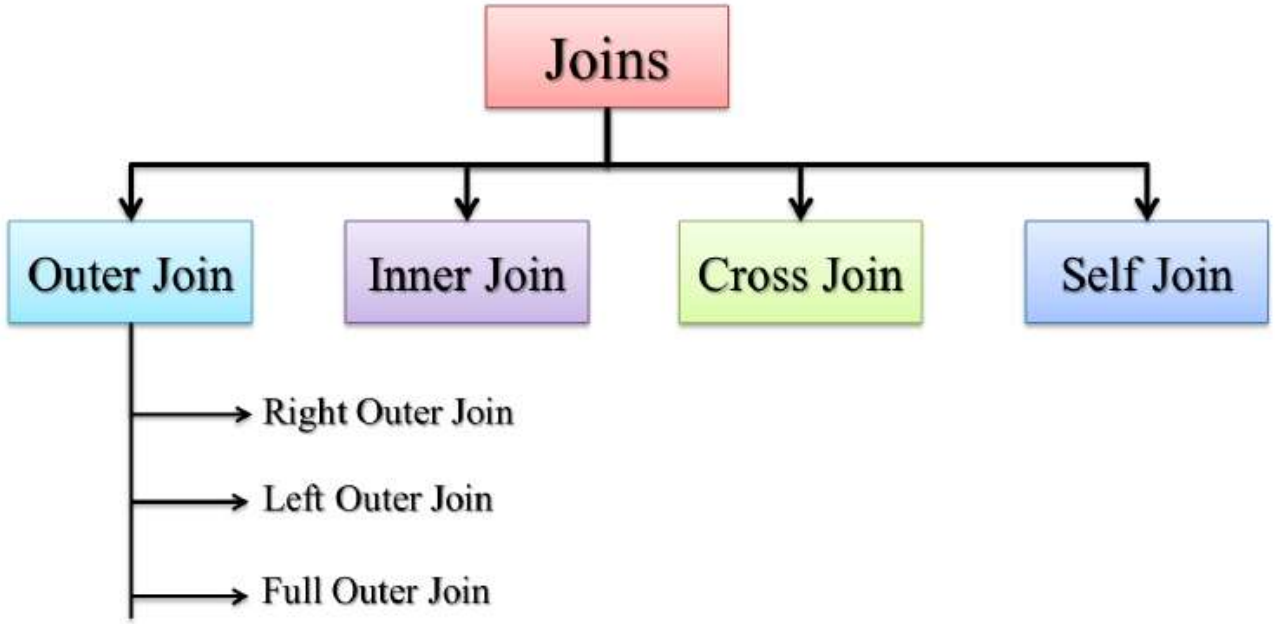
SQL JOIN

- **A Join is a concept that allows us to retrieve data from two or more tables in a single query.**



SQL Joins Types

- There are 6 different types of SQL Server joins:



- **1. INNER JOIN**
- **2. LEFT JOIN**
- **3. RIGHT JOIN**
- **4. FULL JOIN**
- **5. SELF JOIN**
- **6. CARTESIAN JOIN(Cross Join)**

Why SQL JOIN is used?

- If you want to access more than one table through a select statement.
- If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.
- The joining of two or more tables is based on common field between them.
- SQL INNER JOIN also known as simple join is the most common type of join.

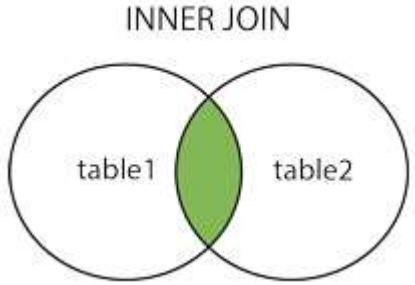
INNER JOIN

- The INNER JOIN keyword selects records that have matching values in both tables.
- **INNER JOIN Syntax**
- **SELECT table1.column1, table2.column2...**

  FROM table1

  INNER JOIN table2

  ON table1.common_field = table2.common_field;



select s.name, s.id, s.gender, s.age, g.grade from StudentDetails as s inner join grade as g on s.id= g.id;

- Create table StudentDetails(Name varchar(50), id int, Age int, Gender varchar(50), Location varchar(50));
- insert into StudentDetails values('sai',1,12, 'female','spain'),('Baba',2,15, 'Male','Newyork'),('ram',3,15, 'Male','spain'),('raghu',4,15, 'female','Newyork'),('Ajay',5,12, 'male','Nijeria'),('Amit',6,12, 'male','spain’);

- create table Grade(Grade varchar(25), id int);
- insert into Grade values('A', 2),('B',3),('A',4),('C',5),('B',7);
- select *from StudentDetails;
- select *from grade;

Inner Join: Get the student details of Fusion with Results who passed in exams

Name	id	Age	Gender	Location
sai	1	12	female	spain
Baba	2	15	Male	Newyork
ram	3	15	Male	spain
raghu	4	15	female	Newyork
Ajay	5	12	male	Nijeria
Amit	6	12	male	spain

StudentDetails

Grade	id
A	2
B	3
A	4
C	5
B	7

Grades

name	id	gender	age	grade
Baba	2	Male	15	A
ram	3	Male	15	B
raghu	4	female	15	A
Ajay	5	male	12	C

Output of Inner Join

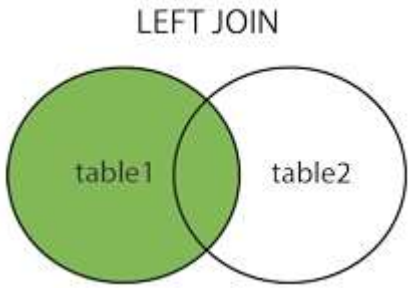
LEFT JOIN

- The LEFT JOIN Keyword returns all records from the left table(table1), and the matching records from the right table(table 2). The result is 0 records from the right side, if there is no match.
- **LEFT JOIN Syntax**
- **SELECT table1.column1, table2.column2...**

  FROM table1

  LEFT JOIN table2

  ON table1.common_field = table2.common_field;



select s.name, s.id, s.gender, s.age, g.grade from StudentDetails as s left join grade as g on s.id= g.id;

Left Join: Get the Student Details of Fusion with Result who appeared in Exam

Name	id	Age	Gender	Location
sai	1	12	female	spain
Baba	2	15	Male	Newyork
ram	3	15	Male	spain
raghu	4	15	female	Newyork
Ajay	5	12	male	Nijeria
Amit	6	12	male	spain

StudentDetails

Grade	id
A	2
B	3
A	4
C	5
B	7

Grade

name	id	gender	age	grade
sai	1	female	12	NULL
Baba	2	Male	15	A
ram	3	Male	15	B
raghu	4	female	15	A
Ajay	5	male	12	C
Amit	6	male	12	NULL

Output of Left Join

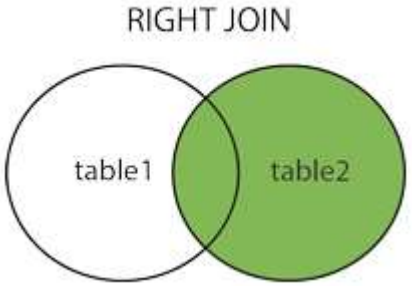
RIGHT JOIN

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.
- **RIGHT JOIN Syntax**
- **SELECT table1.column1, table2.column2...**

  FROM table1

  RIGHT JOIN table2

  ON table1.common_field = table2.common_field;



select s.name, s.id, s.gender, s.age, g.grade from StudentDetails as s right join grade as g on s.id= g.id;

Right Join: Get the All student details with result who passed in exams

Name	id	Age	Gender	Location
sai	1	12	female	spain
Baba	2	15	Male	Newyork
ram	3	15	Male	spain
raghu	4	15	female	Newyork
Ajay	5	12	male	Nijeria
Amit	6	12	male	spain

StudentDetails

Grade	id
A	2
B	3
A	4
C	5
B	7

Grade

name	id	gender	age	grade
Baba	2	Male	15	A
ram	3	Male	15	B
raghu	4	female	15	A
Ajay	5	male	12	C
NULL	NULL	NULL	NULL	B

Output of Right Join

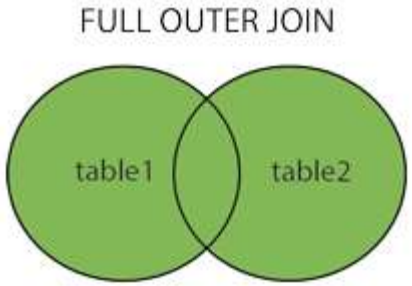
FULL JOIN

- The Full Outer Join keyword returns all records when there is a match in left (table1) or right (table2) table records.
- **Full JOIN Syntax**
- **SELECT table1.column1, table2.column2...**

  FROM table1

  FULL JOIN table2

  ON table1.common_field = table2.common_field;



Full Join

- If your Database does not support FULL JOIN (MySQL does not support FULL JOIN), then you can use **UNION ALL or UNION** clause to combine these two JOINS as shown below.

- select s.name, s.id, s.gender, s.age, g.grade from StudentDetails as s left join grade as g on s.id= g.id

union

- select s.name, s.id, s.gender, s.age, g.grade from StudentDetails as s right join grade as g on s.id= g.id;

name	id	gender	age	grade
sai	1	female	12	NULL
Baba	2	Male	15	A
ram	3	Male	15	B
raghu	4	female	15	A
Ajay	5	male	12	C
Amit	6	male	12	NULL
NULL	NULL	NULL	NULL	B

SELF JOIN

- The SQL SELF JOIN is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- **SELF JOIN Syntax**
- **SELECT a.column_name, b.column_name...**

  FROM table1 a, table1 b

  WHERE a.common_field = b.common_field;

- Que. **To display Employee Name and Their corresponding Manager Name**

- create table Emp(Emp_ID int, Name varchar(255),Mgr_ID int);
- insert into Emp values(1, 'Rahul',3),(2,'Jay',3),(3,'Sonam',4),(4,'Kunal',5),(5,'Ram',6);
- insert into Emp(Emp_ID,Name)values(6,'Rani');
- insert into Emp values(7, 'Veeru',6);
- Select *from Emp;

Emp_ID	Name	Mgr_ID
1	Rahul	3
2	Jay	3
3	Sonam	4
4	Kunal	5
5	Ram	6
6	Rani	NULL
7	Veeru	6

- **SELF Join**
- Select e.name as Emp_Name,m.name Manager from Emp E, Emp M where e.Mgr_ID=M.Emp_id;

Emp_Name	Manager
Rahul	Sonam
Jay	Sonam
Sonam	Kunal
Kunal	Ram
Ram	Rani
Veeru	Rani

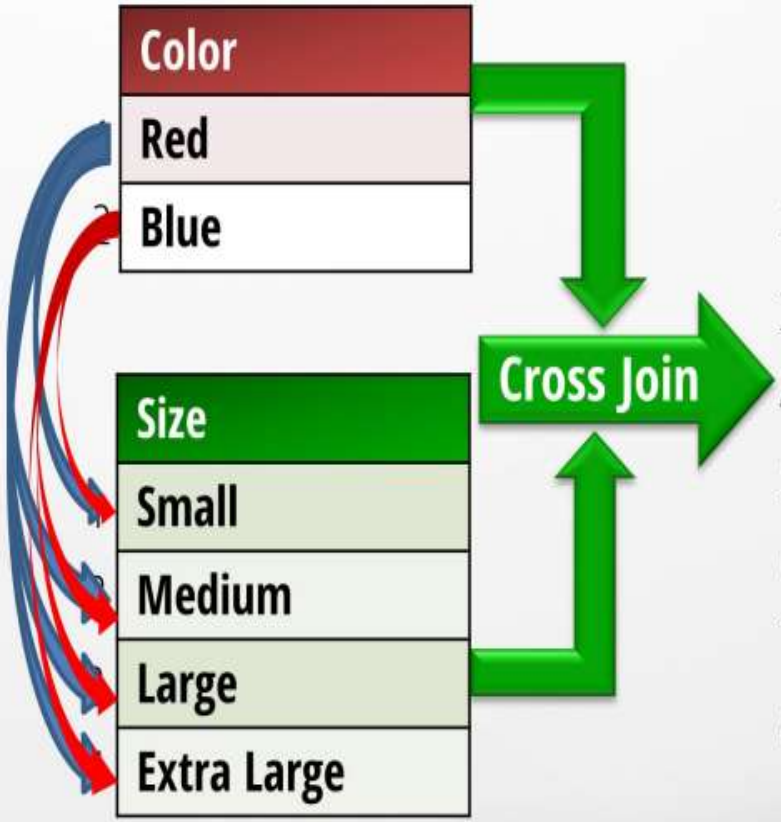
CARTESIAN JOIN

- The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from the two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or where the join-condition is absent from the statement.
- **Syntax**
- **SELECT table1.column1, table2.column2...**

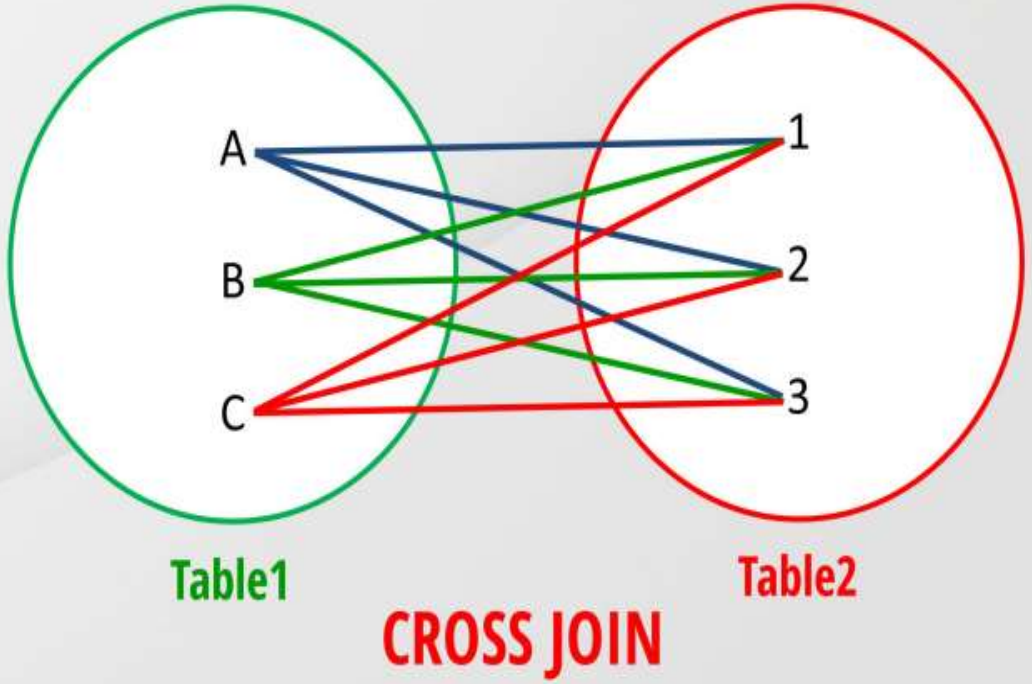
  FROM table1, table2 [, table3]

CROSS JOIN (Cartesian Product & Cross Product)

- ☑ Cross joins are used to return every combination of rows from two tables.



	Color	Size
1	Red	Small
2	Red	Medium
3	Red	Large
4	Red	Extra Large
5	Blue	Small
6	Blue	Medium
7	Blue	Large
8	Blue	Extra Large



- ☑ Cross Join is also called as Cartesian Product or Cross Product.

Examples of Joins- Suppose two tables to performs Join operations

- CREATE TABLE Employee(Employee_ID int, Name varchar(255));
- Insert into Employee values(7369,'Sai');
- Insert into Employee values(7499,'Rahul');
- Insert into Employee values(7521,'Pooja');
- Insert into Employee values(04,'Johnson');
- Select * from Employee;

Employee Table

```
mysql> Select *from Employee;
+-------------+---------+
| Employee_ID | Name    |
+-------------+---------+
|        7369 | Sai     |
|        7499 | Rahul   |
|        7521 | Pooja   |
|           4 | Johnson |
+-------------+---------+
4 rows in set (0.00 sec)
```

- CREATE TABLE Orders(Prod_ID int, Product varchar(255), Employee_ID int);
- insert into Orders values(234, 'Finacle',7369);
- insert into Orders values(657, 'Table',7521);
- insert into Orders values(865, 'QTP',7521);
- Select * from Orders;

• Orders Table

```
mysql> Select *from Orders;
+---------+---------+-------------+
| Prod_ID | Product | Employee_ID |
+---------+---------+-------------+
|     234 | Finacle |        7369 |
|     657 | Table   |        7521 |
|     865 | QTP     |        7521 |
+---------+---------+-------------+
3 rows in set (0.00 sec)
```

CARTESIAN Join

```
mysql> Select Employee.Name, Orders.product from Employee, Orders;
+---------+---------+
| Name    | product |
+---------+---------+
| Sai     | QTP     |
| Sai     | Table   |
| Sai     | Finacle |
| Rahul   | QTP     |
| Rahul   | Table   |
| Rahul   | Finacle |
| Pooja   | QTP     |
| Pooja   | Table   |
| Pooja   | Finacle |
| Johnson | QTP     |
| Johnson | Table   |
| Johnson | Finacle |
+---------+---------+
12 rows in set (0.00 sec)
```

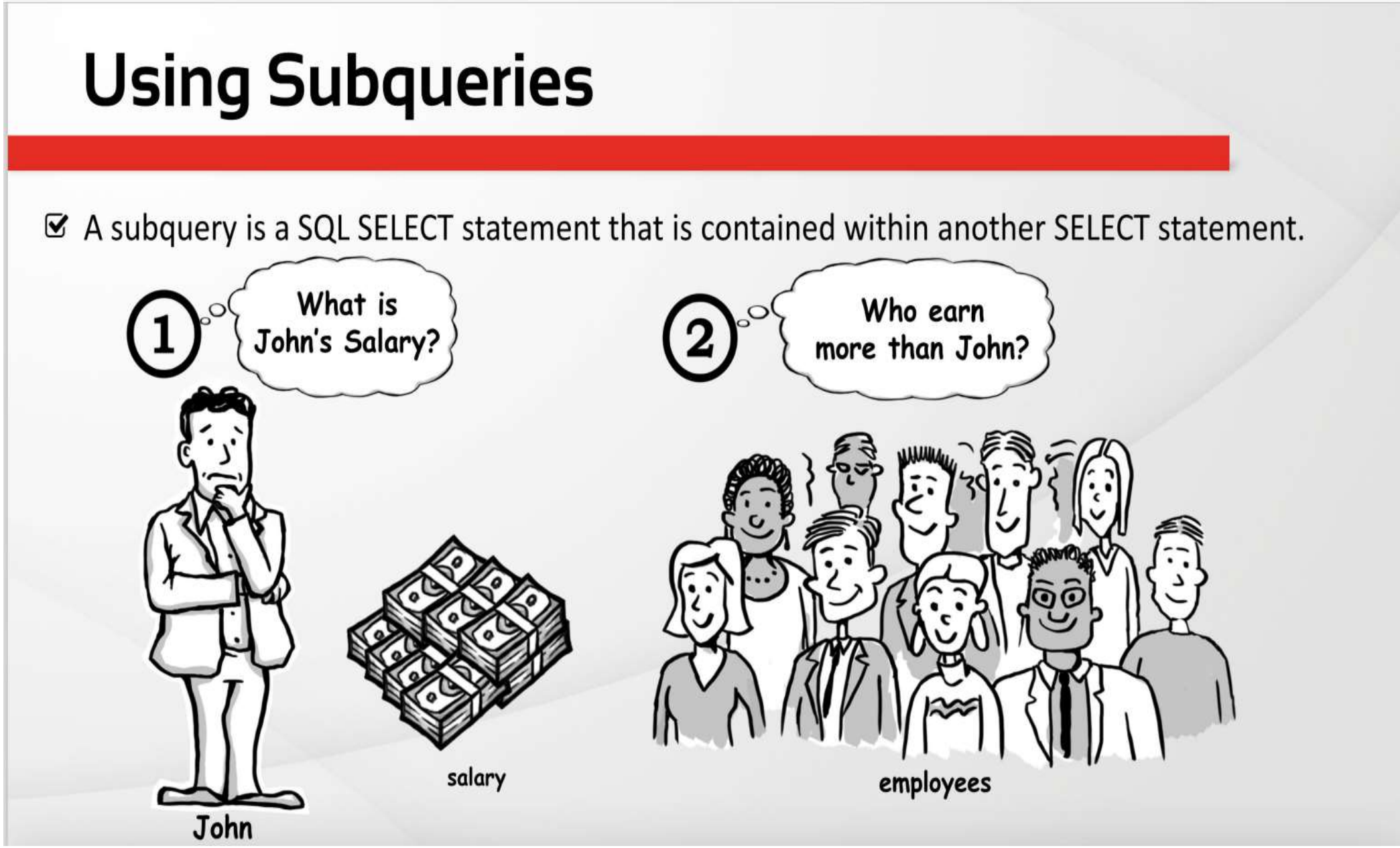
SQL Sub Queries

- **A MySQL subquery is a query nested within another query such as SELECT, INSERT, UPDATE or DELETE.**
- Also, a subquery can be nested within another subquery.
- **A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query.**
- The output of inner query becomes input of outer query

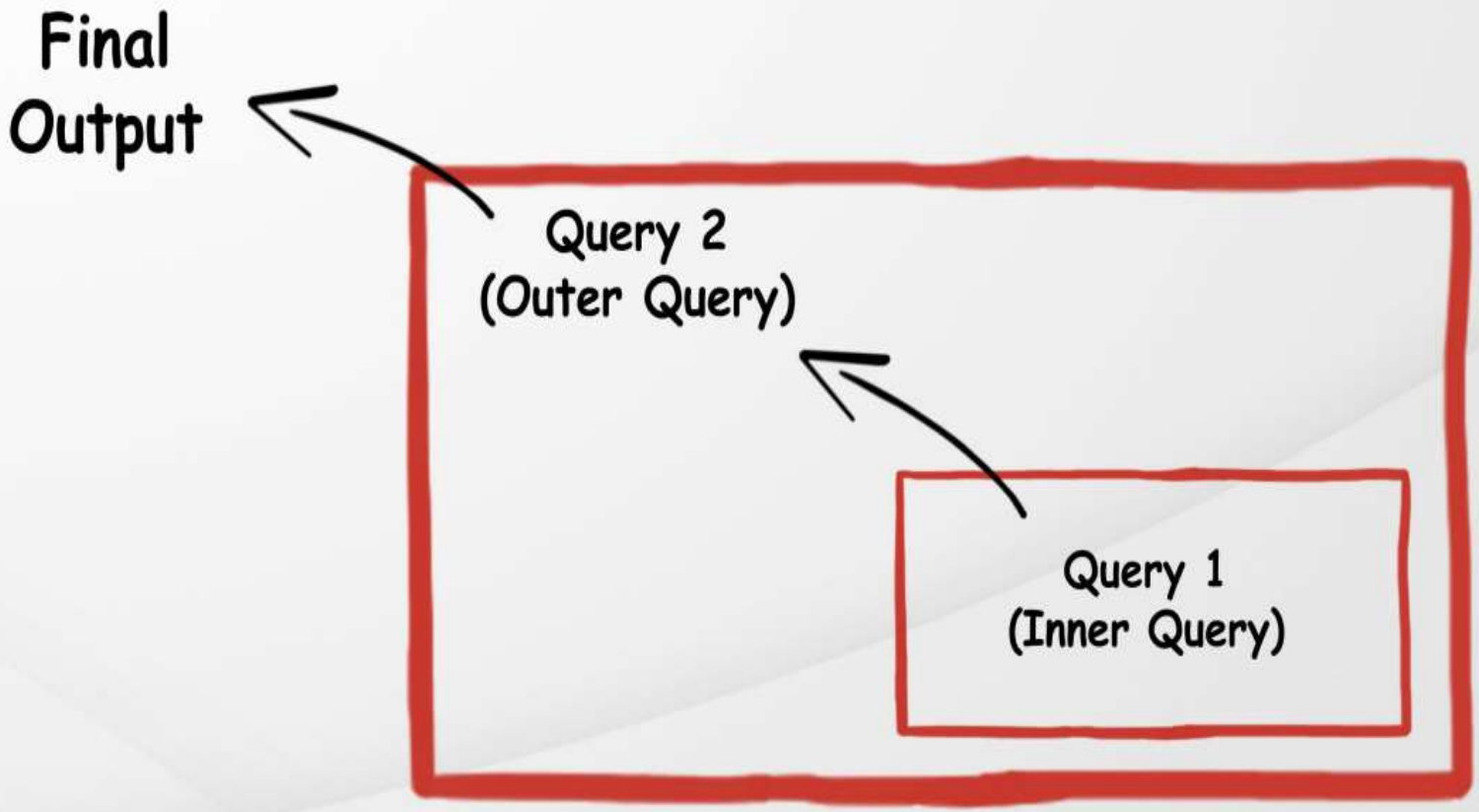
Types of sub queries;

- 1. Single row sub query (<=, >=, !=)
- 2. Multi row sub Query (IN, ANY, ALL)

- MySQL Subquery Syntax



Using a Subquery (Working Mechanism)



Using a Subquery (Working Mechanism)

```
SELECT column_names FROM table
      WHERE expression-operator
            (SELECT column_names FROM table);
```

```
SELECT first_name, last_name, salary
FROM employees
WHERE salary >
            (SELECT salary
             FROM employees
             WHERE employee_id = 145);
```

OUTPUT

	FIRST_NAME	LAST_NAME	SALARY
1	Steven	King	24000
2	Neena	Kochhar	17000
3	Lex	De Haan	17000

SQL Sub Queries

- select employees whose is greater than 'Michael's' salary
- select * from employees where salary > (select salary from employees where first_name ='Michael');
- 2nd highest salary from employees table
- select max(salary) from employees where salary < (select max(salary) from employees);
- List employees having highest salary
- select first_name from employees where salary = (select max(salary) from employees);
- 3rd highest salary from employees table
- select max(salary) from employees where salary <

  (select max(salary) from employees where salary <

  (select max(salary) from employees));

SQL Sub Queries

Query to get first name, lastname department id and department name

- **select first_name, last_name, department_id,**

(select department_name from departments where employees.department_id = departments.department_id) departmentname from employees;

Find the customers who have not placed any orders

- **SELECT customerName FROM customers WHERE customerNumber NOT IN (SELECT DISTINCT customerNumber FROM orders);**

SQL Constraints

- It specify the rule to user during the data insertion into the table

- **1.NOT NULL**
- **2.UNIQUE**
- **3.Primary key**
- **4.Foreign key**
- **5.Check constraint**
- **6.Default constraint**

Not Null Constraint

- 1.It does not allow to user insert the null value
- 2.If we want to make any field as mandatory then we will use NOT NULL constraint
- Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

Example of Not Null Constraint

- Create table customer(ID int **NOT NULL**, NAME Varchar(25),Lname varchar(30),Aadhar int);

- Insert into customer (ID,NAME,LNAME,AADHAR)Values(1,'Asif','Shaikh',2525); -- Correct

- Insert into customer (NAME,LNAME,AADHAR)Values('Asif','Shaikh',2525); -- Error

Unique Constraint

- 1.It does not accept duplicate value
- 2.It does accept only UNIQUE value

- create table student1(sno int **Unique**, sname varchar(10), marks int);

- insert into student1 values(101,'arun',50);
- insert into student1 values(101,'kiran',60); -- 101 not allowed
- insert into student1 values(null,'suresh',80);
- insert into student1 values(null,'raj',60);

* **Unique constraint column can accept multiple NULLS.**

Default Constraint

- It is used to set the default value
- **Example**
- Create table EMP(ID int, Name varchar(25),City varchar(30) Default ‘Satara');

- insert into EMP(ID,Name, city)Values(1,'Vishal','Pune');

- Insert into EMP(ID,Name)values(2,'Vinay');

Check Constraint

- Check constraint is used for allowing to user to enter specific values into column.

Example

- create table student(sno int, sname varchar(15),

marks int **check**(marks between 50 and 100));

- insert into student values(101,'amith',60); # valid
- insert into student values(101,'amith',45); # in valid
- insert into student values(101,'amith',105); #invalid

Primary Key Constraint

- 1.Primary key is a combination of NOT NULL + UNIQUE Key
- 2.It does not allow duplicate values
- 3.It does not allow NULL values
- 4.in each table only one primary key
- **Example**
- create table student1(sno int **primary key**, sname varchar(20), marks int);
- insert into student1 values(101,'arun',50); -- right
- insert into student1 values(101,'suresh',60); -- Invalid
- insert into student1 values(null,'suresh',60); -- Invalid

Auto Increment Keyword

- Sometimes while creating a table we do not have unique identifier within the table hence we face difficulty in choosing Primary Key. so as to resolve such an issue we've to manually provide unique keys to every record but this is often also a tedious task. So we can use Auto Increment feature that automatically generates a numerical Primary key value for every new record inserted.

- CREATE TABLE Students(Student_ID int **AUTO_INCREMENT** PRIMARY KEY, First_Name varchar(255),Last_Name varchar(255));
- INSERT INTO Students(First_Name, Last_Name) VALUES ('Anish', 'Jain');
- INSERT INTO Students(First_Name, Last_Name) VALUES ('Akshita', 'Sharma');
- INSERT INTO Students(First_Name, Last_Name) VALUES ('Shruti', 'Sogani');
- insert into students(first_name, last_name) values('nnn','nnkkk');
- select *from students;

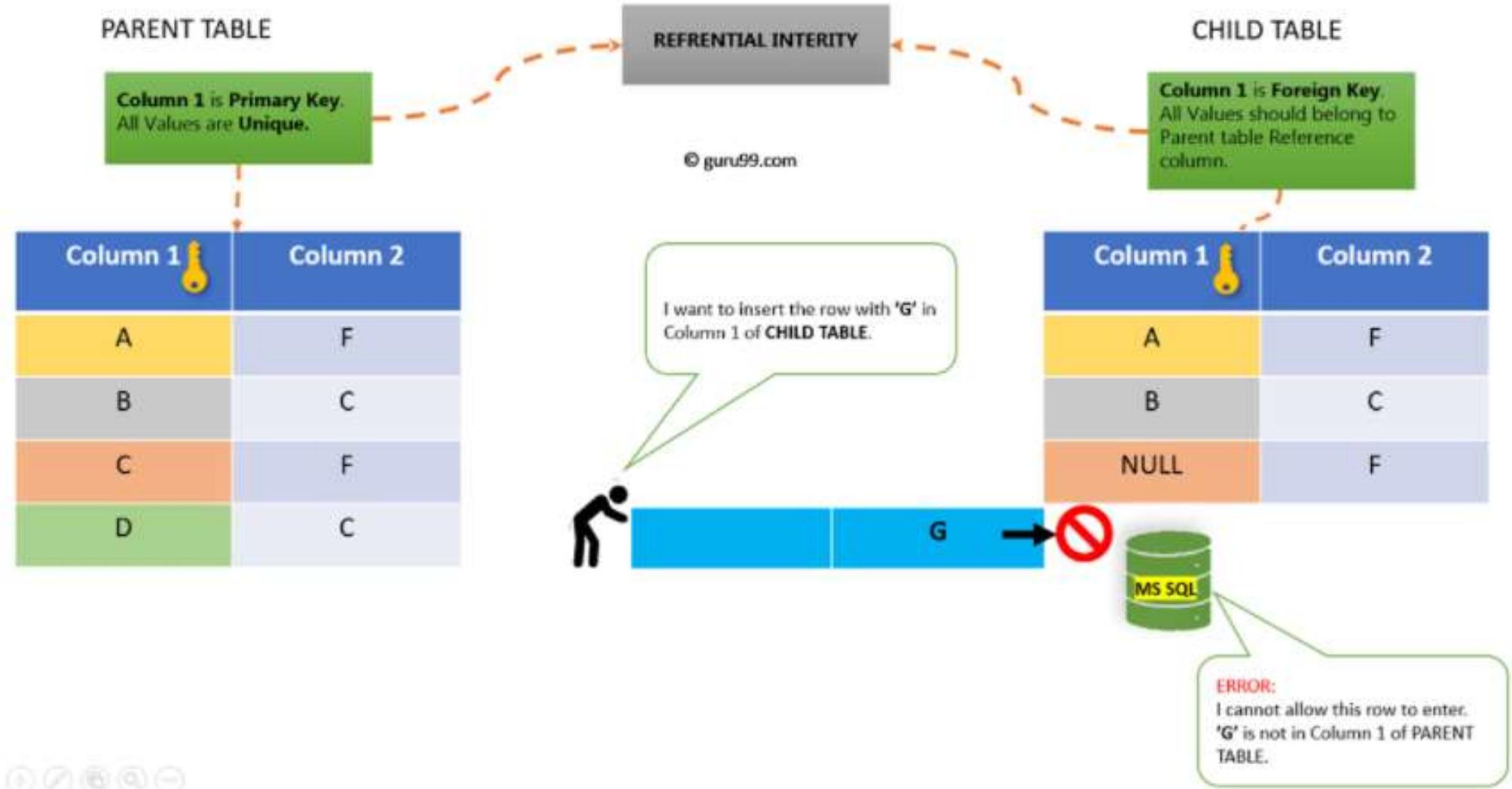
Student_ID	First_Name	Last_Name
1	Anish	Jain
2	Akshita	Sharma
3	Shruti	Sogani
4	nnn	nnkkk

Foreign Key Constraint

A **Foreign Key** provides a way of enforcing referential integrity within SQL Server. In simple words, foreign key ensures values in one table must be present in another table.

- **Rules for FOREIGN KEY**
- NULL is allowed in SQL Foreign key.
- The table being referenced is called the Parent Table
- The table with the Foreign Key in SQL is called Child Table.
- The SQL Foreign Key in child table references the primary key in the parent table.
- This parent-child relationship enforces the rule which is known as “Referential Integrity.”

The Below Foreign Key in SQL example with diagram summarizes all the above points for FOREIGN KEY



How Foreign Key Works

- FOREIGN KEY CONTRAINT (OR) REFERENCIAL INTEGRITY
- // parent Table

- create table school(sno int, sname varchar(15), marks int, primary key(sno));
- insert into school values(101,'arun',90);
- insert into school values(102,'kiran',70);
- insert into school values(103,'amit',80);
- Select * from school;

- // child table
- create table library(sno int, foreign key(sno)references school(sno), book_name varchar(10));
- insert into library values(102,'java'); -- valid
- insert into library values(108,'c'); -- invalid
- insert into library values(null,'dotnet'); -- valid

On Delete cascade keyword

- We can delete the rows from the parent table and the corresponding child table rows deleted automatically.

-- **Parent table [having Primary Key]**

- create table school(sno int Primary Key, sname varchar(15), marks int);
- insert into school values(101,'arun',90);
- insert into school values(102,'kiran',70);
- insert into school values(103,'amit',80);
- Select * from school;

-- Child table [having foreign key]

- create table library(sno int, foreign key(sno)references school(sno) **on delete cascade**, book_name varchar(255));
- insert into library values(101, 'Java'); -- valid
- insert into library values(444,'C programming'); -- Invalid becoz 444 is not in parent table
- insert into library values(null,'html'); -- valid
- select *from library;

-- delete

- delete from school where sno=101;

On Update cascade keyword

- When we create a foreign key using UPDATE CASCADE the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key

```
CREATE TABLE student (
  id INT PRIMARY KEY,
  courseID INT,
  FOREIGN KEY(courseID) REFERENCES course(id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
);
```

Primary Key	Foreign Key
A primary key is used to ensure data in the specific column is unique.	A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables.
It does not allow NULL values.	It can also contain NULL values.
Only one primary key is allowed in a table.	Whereas more than one foreign key is allowed in a table.
It is a combination of UNIQUE and Not Null constraints.	It can contain duplicate values and a table in a relational database.

How to Create a Table With Multiple Foreign Keys in SQL?

```
CREATE TABLE student_details(
  stu_id VARCHAR(8) NOT NULL PRIMARY KEY,
  stu_name VARCHAR(20),
  stu_branch VARCHAR(20) FOREIGN KEY REFERENCES student_branch_details(stu_branch),
  stu_pin_code VARCHAR(6) FOREIGN KEY REFERENCES student_address(stu_pin_code)
  );
CREATE TABLE student_branch_details(
  stu_branch VARCHAR(20) PRIMARY KEY,
  subjects INT,
  credits INT
);
CREATE TABLE student_address(
  stu_pin_code VARCHAR(6) PRIMARY KEY,
  stu_state VARCHAR(20),
  student_city VARCHAR(20)
);
```

View

- A view is a virtual table whose contents are defined by a query.
- Uses-
- 1) Restrict access to the database because the view can display a selective portion of the database.
- 2) Provide group of users access to data according to their particular criteria.
- Syantax

The basic **CREATE VIEW** syntax is as follows −

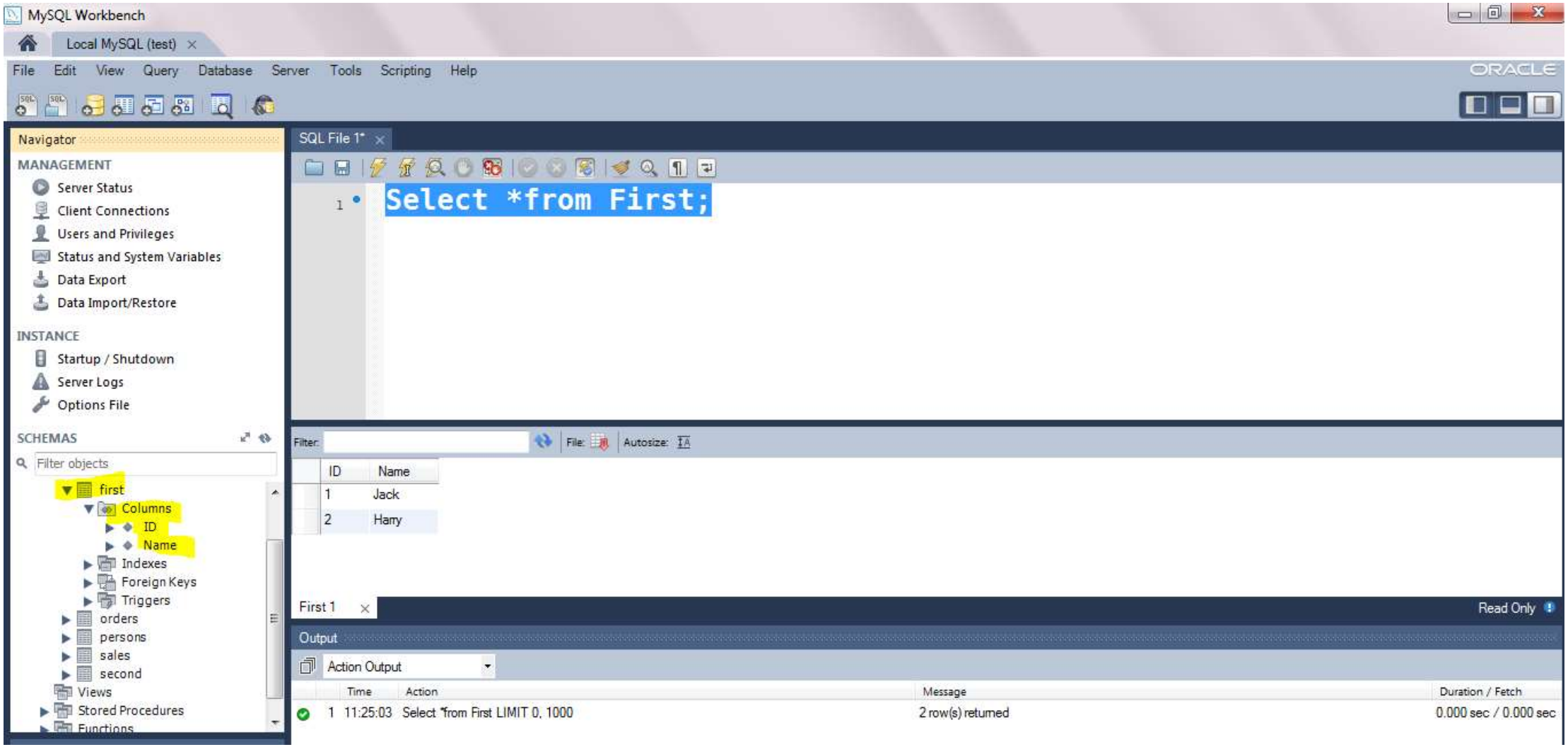
```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

- Eg.
- Suppose “First” is the table already available in database. Creating view use below query.

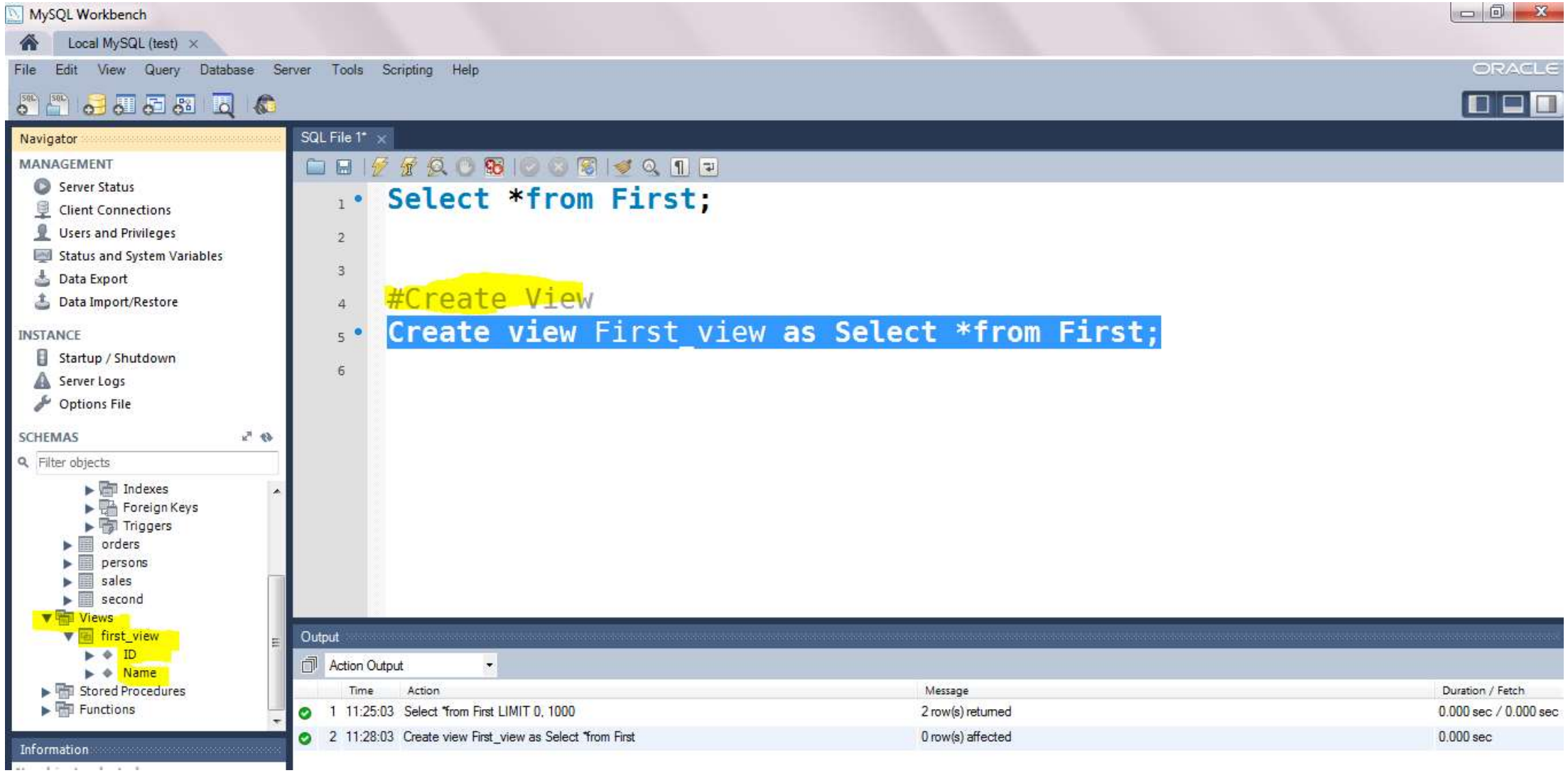
```
Create view First_view as Select *from First;

Select *from first_view;
```

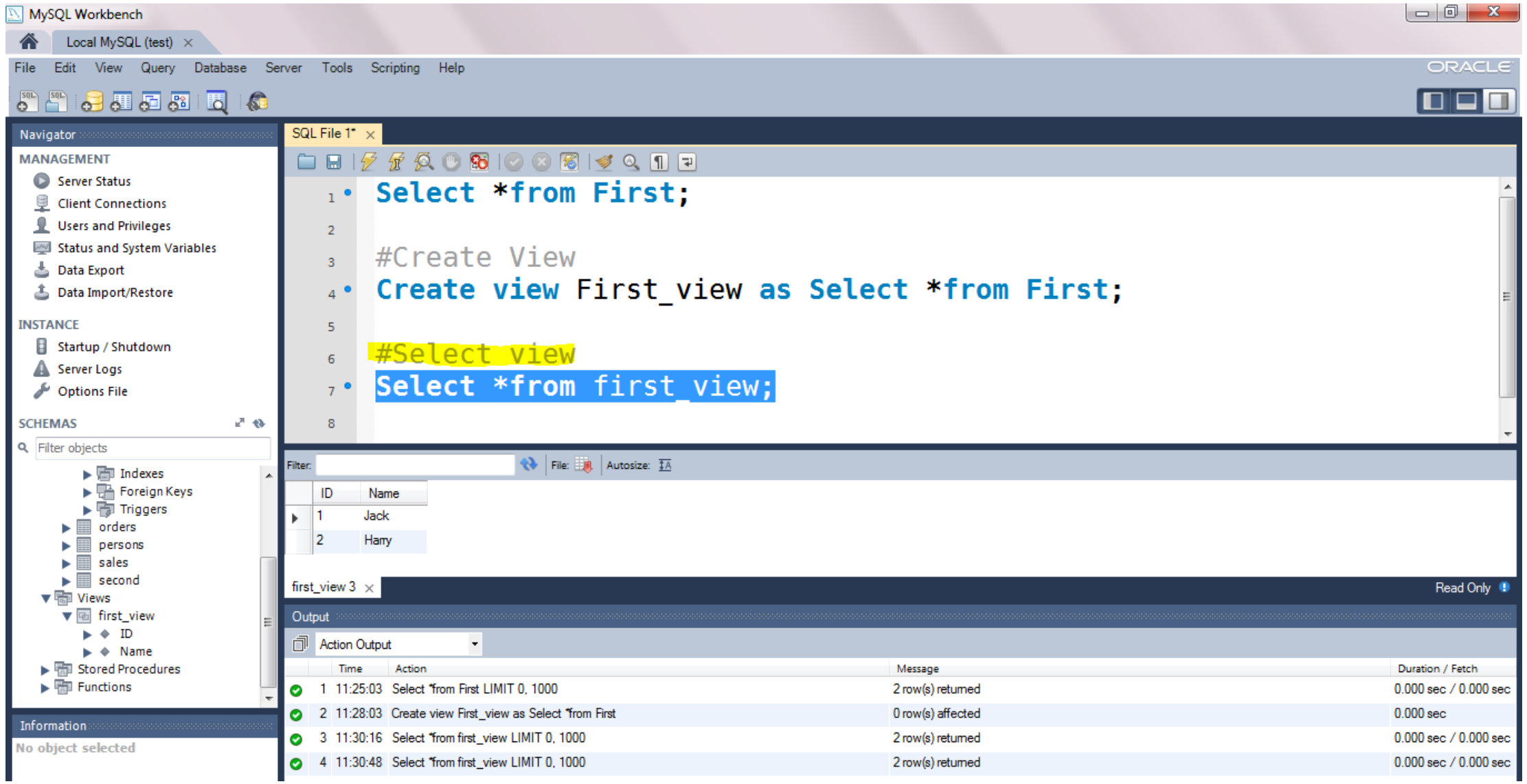
For drop view, use below query

```
Drop view first_view;
```


MySQL Workbench
Local MySQL (test)
File Edit View Query Database Server Tools Scripting Help
ORACLE
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
SCHEMAS
Filter objects
first
Columns
ID
Name
Indexes
Foreign Keys
Triggers
orders
persons
sales
second
Views
Stored Procedures
Functions
SQL File 1*
1 Select *from First;
Filter:
File:
Autosize:
ID Name
1 Jack
2 Harry
First 1
Read Only
Output
Action Output
Time Action Message Duration / Fetch
1 11:25:03 Select *from First LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec

View


MySQL Workbench
Local MySQL (test)
File Edit View Query Database Server Tools Scripting Help
ORACLE
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
SCHEMAS
Filter objects
Indexes
Foreign Keys
Triggers
orders
persons
sales
second
Views
first_view
ID
Name
Stored Procedures
Functions
Information
SQL File 1*
1 Select *from First;
2
3
4 #Create View
5 Create view First_view as Select *from First;
6
Output
Action Output
Time Action Message Duration / Fetch
1 11:25:03 Select *from First LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
2 11:28:03 Create view First_view as Select *from First 0 row(s) affected 0.000 sec


MySQL Workbench
Local MySQL (test)
File Edit View Query Database Server Tools Scripting Help
ORACLE
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
SCHEMAS
Filter objects
Indexes
Foreign Keys
Triggers
orders
persons
sales
second
Views
first_view
ID
Name
Stored Procedures
Functions
Information
No object selected
SQL File 1*
1 Select *from First;
2
3 #Create View
4 Create view First_view as Select *from First;
5
6 #Select view
7 Select *from first_view;
8
Filter:
File:
Autosize:
ID Name
1 Jack
2 Harry
first_view 3
Read Only
Output
Action Output
Time Action Message Duration / Fetch
1 11:25:03 Select *from First LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
2 11:28:03 Create view First_view as Select *from First 0 row(s) affected 0.000 sec
3 11:30:16 Select *from first_view LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
4 11:30:48 Select *from first_view LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec

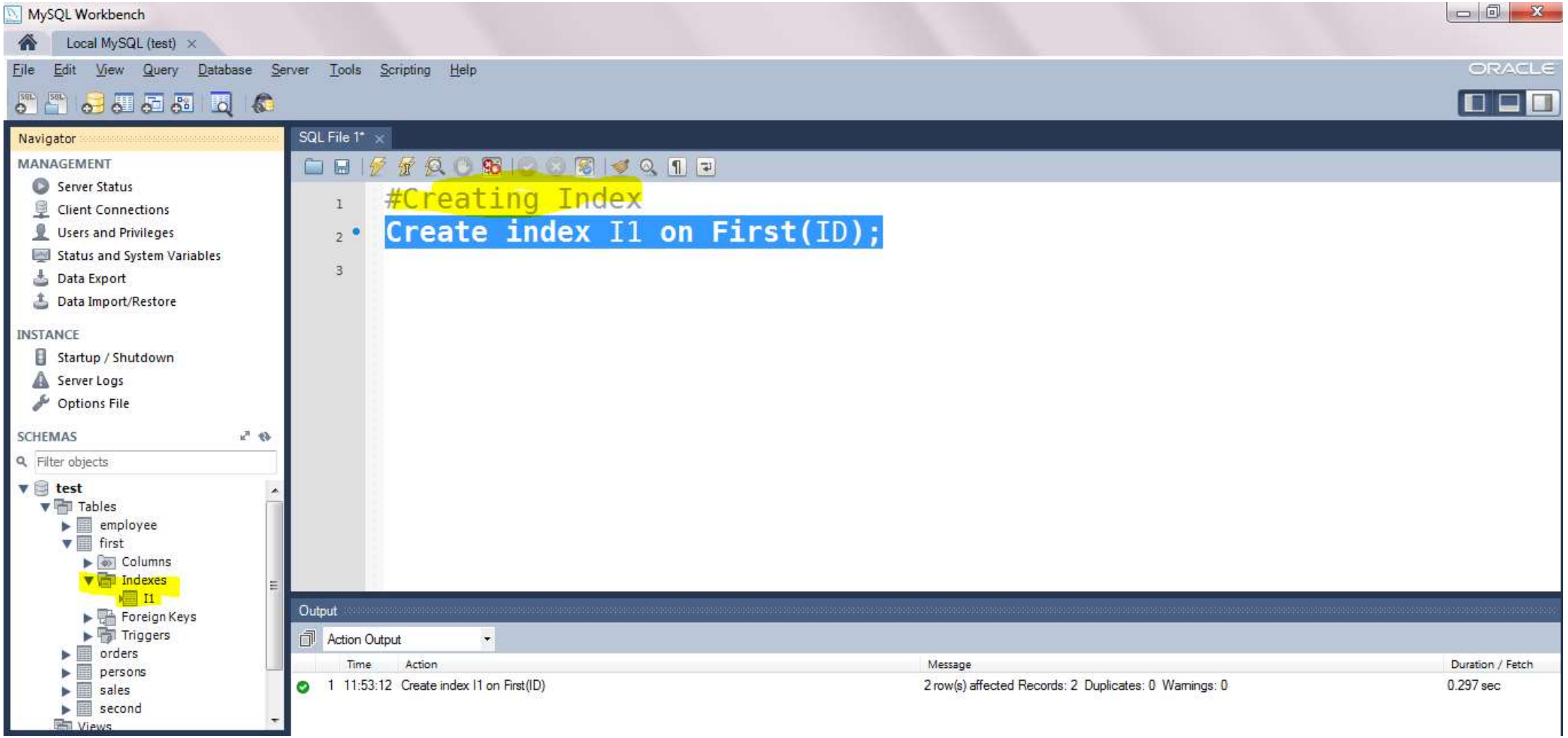
Index

- Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.
- **Syntax**

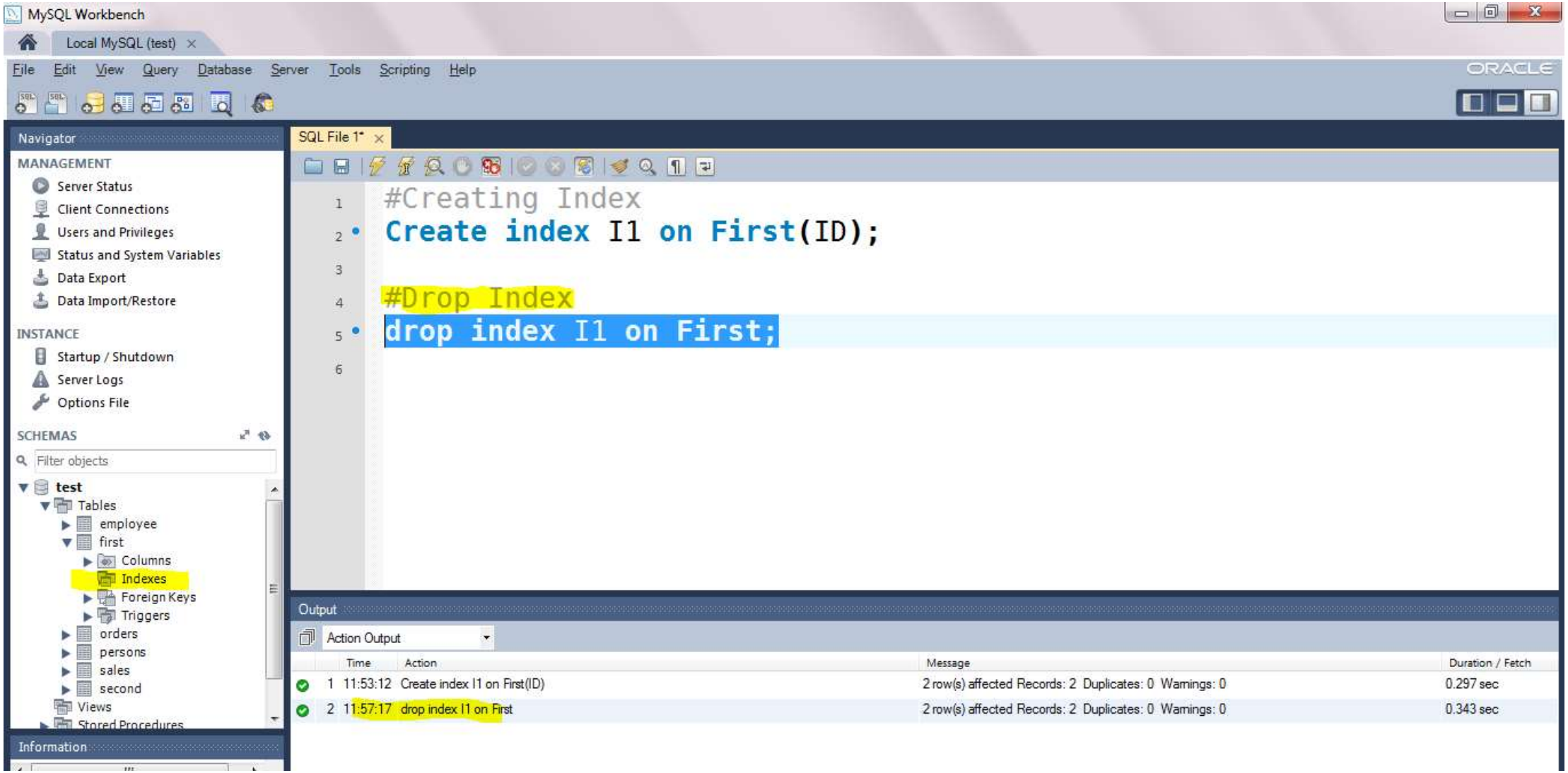
```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

```
DROP INDEX index_name ON table_name;
```

Create Index



Drop index


MySQL Workbench
Local MySQL (test)
File Edit View Query Database Server Tools Scripting Help
ORACLE
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
SCHEMAS
Filter objects
test
Tables
employee
first
Columns
Indexes
Foreign Keys
Triggers
orders
persons
sales
second
Views
Stored Procedures
Information
SQL File 1*
1 #Creating Index
2 Create index I1 on First(ID);
3
4 #Drop Index
5 drop index I1 on First;
6
Output
Action Output
Time Action Message Duration / Fetch
1 11:53:12 Create index I1 on First(ID) 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 0.297 sec
2 11:57:17 drop index I1 on First 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 0.343 sec

Store Procedures

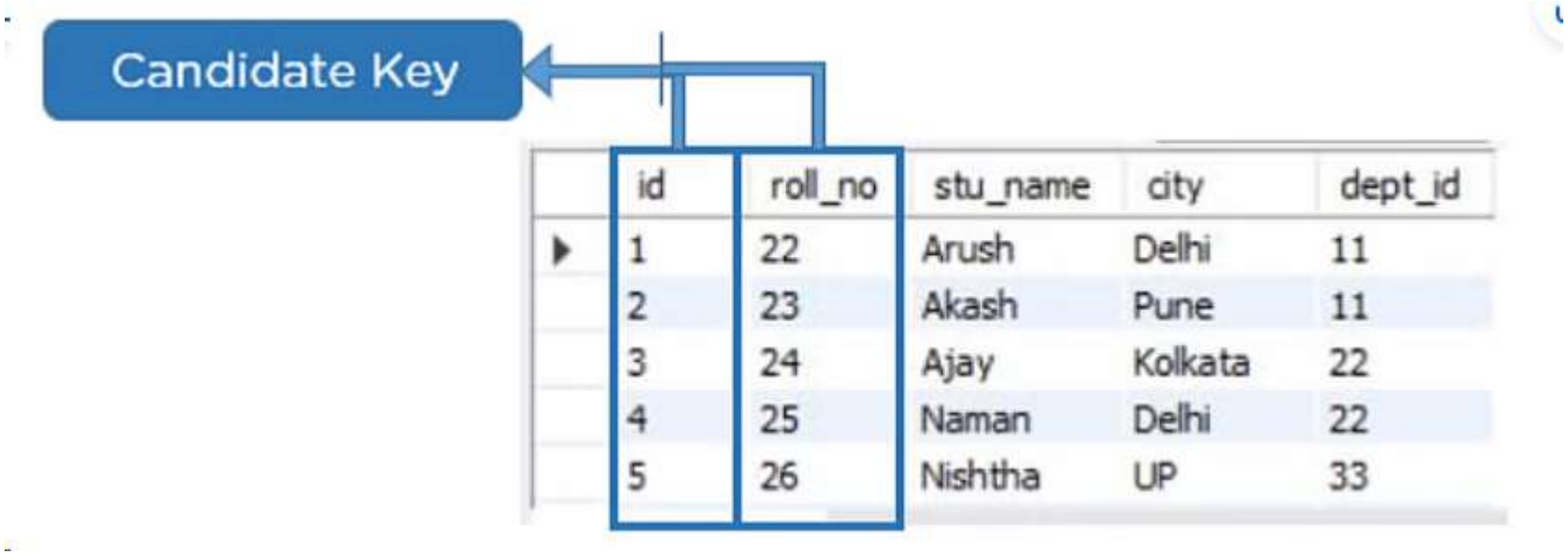
A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Candidate Key

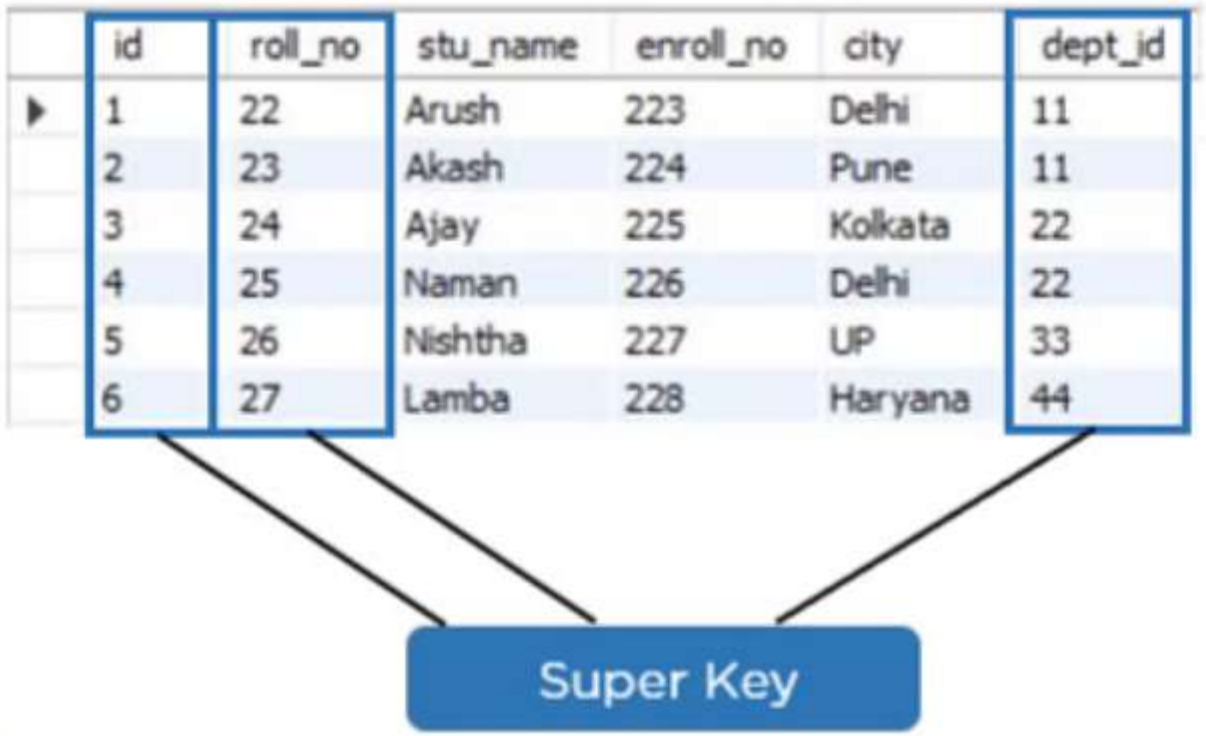
- A candidate key is a set of one or more columns that can identify a record uniquely in a table, and YOU can use each candidate key as a Primary Key.
- Now, let's use an example to understand this better.



id	roll_no	stu_name	city	dept_id
1	22	Arush	Delhi	11
2	23	Akash	Pune	11
3	24	Ajay	Kolkata	22
4	25	Naman	Delhi	22
5	26	Nishtha	UP	33

Super Key

- Super key is a set of over one key that can identify a record uniquely in a table, and the Primary Key is a subset of Super Key.
- Let's understand this with the help of an example.



Normalization in SQL

- In SQL, the process of structuring and resolving data in a database is known as normalization. It is achieved by applying a set of rules and procedures to a database, known as normalization forms.

- There are several normalization forms, including the First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-Codd Normal Form (BCNF), and the Fourth Normal Form (4NF). Each form builds on the previous one, providing a set of guidelines for organizing data in a way that meets the specific requirements of a database.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE

First Normal Form (1NF)

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key
- **Example:** Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

- In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.
- To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

- **SQL Questions**
- 1. Explain DML and DDL?
- 2. How many Aggregate functions are available in SQL?
- 3. What is the difference in BETWEEN and IN condition operators?
- 4. What is the difference between the HAVING clause and WHERE clause?
- 5. What is the difference between DELETE, TRUNCATE & DROP ?
- 6. What are different Clauses used in SQL?
- 7. What are different SQL constraints?
- 8. What is the difference between UNIQUE key, PRIMARY KEY & FORGIN KEY constraints?
- 9. What are different JOINS used in SQL?

- 10. How to write a query to show the details of a student from Students table whose name start with K?
- 11.What is the syntax to add a record to a table?
- 12. What is the syntax of GROUP BY in SQL?
- 13. Define the SQL DELETE statement.
- 14. Write a SQL SELECT query that only returns each name only once from a table?
- 15. Write an SQL query to get the first maximum salary of an employee from a table named employee_table.
- 16. Write an SQL query to get the second maximum salary of an employee from a table named employee_table.
- 17. Write an SQL query to get the third maximum salary of an employee from a table named employee_table.

- 18. Write an SQL query to fetch unique values from a table?
- 19. Write an SQL query to fetch data from table whose name start with Vipul & Krishna?
- 20 Write an SQL query to print details of the Workers whose SALARY lies between 100000 and 500000.
- 21. Write an SQL query to print details of the Workers who have joined in Feb’2014.
- 22. Write an SQL query to fetch the count of employees working in the department ‘Admin’.
- 23. Write an SQL query to fetch worker names with salaries >= 50000 and <= 100000.
- 24. What are the Indexes & Views in SQL?
- 25. What is schema?

For practice see SQL quires in below link

- https://www.techbeamers.com/sql-query-questions-answers-for-practice/

Thank You

Best of Luck