SQL PROJECT

EMPLOYERS DETAIL

BY

SABARI NATHAN.K

Intro to SQL

- SQL Stands for Structured Query Language.
- It Is Used to Interact With the Relational Databases.
- It Is a Domain Specific Language.
- Designed For Managing And Querying Structured Data In Relational Database Management System(RDBMS).

INTRO TO MYSQL

- MySQL is an open-source relational database management system(RDBMS).
- Developed By MySQL AB.
- Its is known for its reliability, scalability and performance.
- MySQL Fully Supports SQL.
- MySQL Supports Stored procedures and Triggers.

RELATIONAL DATABASES

- Oracle Database
- MySQL
- Microsoft SQL Server
- PostgreSQL
- IBM Db2
- SQLite

What is Database

- A database is an organized collection of structured data or information, typically stored electronically in a computer system. It is designed to efficiently manage, retrieve, update, and manipulate data according to the needs of various applications and users.

DBMS

- It is a software that enables users to interact with the database.
- It provides an interface for creating, modifying, and querying the database, as well as tools for managing data integrity, security, and performance.

RDBMS

- RDBMS stands for Relational Database Management System. It's a type of database management system (DBMS) that organizes data into tables, which can be related to each other based on common fields.
- RDBMS follows the relational model of data storage and retrieval, which was proposed by E.F. Codd in the 1970s.

Data Types

- A Data Type is an attribute that specifies the type of data that can be stored in a column of a database table.
- Data types in MySQL help ensure data integrity, optimize storage space, and facilitate efficient querying and manipulation of data.

Types of DataTypes

- Numeric DataType
- Character Data Type
- Date and Time Data Type
- Boolean Data Type
- Binary Data Type

NUMERIC DATA TYPE

DATA TYPE SYNTAX	DESCRIPTION
TINYINT	A small integer data type that typically occupies 1 byte of storage. Suitable for storing small integers ranging from -128 to 127 (signed) or 0 to 255 (unsigned).
SMALLINT	A small integer data type that usually occupies 2 bytes of storage. Suitable for storing integers ranging from -32,768 to 32,767 (signed) or 0 to 65,535 (unsigned).
MEDIUMINT	An integer data type that typically occupies 3 bytes of storage. Suitable for storing integers ranging from -8,388,608 to 8,388,607 (signed) or 0 to 16,777,215 (unsigned).
INT	An integer data type that typically occupies 4 bytes of storage. Suitable for storing integers ranging from -2,147,483,648 to 2,147,483,647 (signed) or 0 to 4,294,967,295 (unsigned).
BIGINT	A large integer data type that typically occupies 8 bytes of storage. Suitable for storing large integers ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (signed) or 0 to 18,446,744,073,709,551,615 (unsigned).
FLOAT(m,d)	A floating-point numeric data type used for representing approximate real values. 'm' represents the total number of digits stored, and 'd' represents the number of digits after the decimal point.
DOUBLE(m,d)	A double-precision floating-point numeric data type for representing approximate real values with higher precision compared to FLOAT. 'm' represents the total number of digits stored, and 'd' represents the number of digits after the decimal point.
DECIMAL(m,d)	A fixed-point numeric data type used for representing exact numeric values. 'm' represents the total number of digits stored, and 'd' represents the number of digits after the decimal point.
BIT(m)	A data type used for storing bit values. 'm' represents the number of bits used, which can range from 1 to 64.
BOOLEAN	A data type used for representing truth values, typically stored as 1 (true) or 0 (false).

DATE AND TIME DATATYPES

Data Type Syntax	Maximum Size	Explanation
YEAR[(2|4)]	Year value as 2 digits or 4 digits.	The default is 4 digits. It takes 1 byte for storage.
DATE	Values range from '1000-01-01' to '9999-12-31'.	Displayed as 'yyyy-mm-dd'. It takes 3 bytes for storage.
TIME	Values range from '-838:59:59' to '838:59:59'.	Displayed as 'HH:MM:SS'. It takes 3 bytes plus fractional seconds for storage.
DATETIME	Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.	Displayed as 'yyyy-mm-dd hh:mm:ss'. It takes 5 bytes plus fractional seconds for storage.
TIMESTAMP(m)	Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' TC.	Displayed as 'YYYY-MM-DD HH:MM:SS'. It takes 4 bytes plus fractional seconds for storage.

STRING DATA TYPES

Data Type Syntax	Explanation
CHAR(size)	Fixed-length character string data type. Stores a specific number of characters, padded with spaces if necessary. Ideal for fields with consistent lengths, such as postal codes.
VARCHAR(size)	Variable-length character string data type. Stores variable-length strings up to a specified maximum size. Efficient for fields with varying lengths, like names or addresses.
TINYTEXT(size)	Variable-length text data type with a maximum size of 255 characters. Suitable for short, variable-length textual data
TEXT(size)	Variable-length text data type with a maximum size of 65,535 characters. Appropriate for longer text content such as descriptions or comments.
MEDIUMTEXT(size)	Variable-length text data type with a maximum size of 16,777,215 characters. Used for storing longer text fields like articles or blog posts.
LONGTEXT(size)	Variable-length text data type with a maximum size of 4,294,967,295 characters. Suitable for storing extremely long textual data, such as novels or large documents.
BINARY(size)	Fixed-length binary string data type. Stores binary strings with a specific length, padded with zeros if necessary. Useful for storing binary data such as images or files.
VARBINARY(size)	Variable-length binary string data type. Stores variable-length binary data up to a specified maximum size. Efficient for storing variable-length binary data like images or documents.
ENUM	Enumeration data type. Defines a set of predefined values that a column can contain. Useful for fields that have a limited set of possible values, like gender or status.
SET	Set data type. Similar to ENUM but allows for multiple values to be selected from a predefined set. Suitable for fields where multiple options can be chosen simultaneously, such as interests or skills.

TYPES OF SQL COMMANDS

- DML – Data Manipulation Language.
- DDL – Data Definition Language.
- DCL – Data Control Language.
- TCL – Transaction Control Language.
- DQL – Data Query Language.

DATA MANIPULATION LANGUAGE

- **DML commands are used to manipulate data stored in the database tables.**

- **SELECT**: Retrieves data from one or more tables.
- **INSERT**: Adds new rows of data into a table.
- **UPDATE**: Modifies existing data in a table.
- **DELETE**: Removes rows of data from a table.

Data Definition Language

- **DDL commands are used to define, modify, and delete database objects such as tables, indexes, and views.**
- **CREATE**: Creates new database objects like tables, indexes, or views.
- **ALTER**: Modifies the structure of existing database objects.
- **DROP**: Deletes existing database objects.
- **TRUNCATE**: Removes all data from a table, but keeps the table structure intact.

Data Control Language

- **DCL commands are used to control access to data within the database.**
- **GRANT**: Gives specific privileges to database users.
- **REVOKE**: Removes specific privileges from database users.

TRANSACTION CONTROL LANGUAGE

- **Transaction control language are used to manage the changes made to the database.**

- **COMMIT**: Saves changes made during the current transaction.
- **ROLLBACK**: Reverts the database to its state before the start of the current transaction.
- **SAVEPOINT**: Sets a point within a transaction to which you can later roll back.

DATA QUERY LANGUAGE

- DQL commands are used to retrieve data from the database.
- The primary DQL command is **SELECT**, which is used to fetch data based on specified criteria.

SQL CONSTRAINTS

- **PRIMARY KEY -** Ensures uniqueness of each row in a table.
- **FOREIGN KEY** - Enforces referential integrity by ensuring values in a column match values in another table's primary key.
- **UNIQUE** - Ensures that all values in a column are unique.
- **CHECK -** Validates data based on a condition.
- **NOT NULL** - Ensures that a column cannot have NULL values.
- **DEFAULT -** Provides a default value for a column when no value is specified.
- **INDEX** - Improves the performance of queries by allowing fast lookup of data.

DATABASE CREATION

- **SYNTAX:**

  Create database database_name;

- **Query:**

```
1 •  Create database employee_det;
```

- **Output:**

Action Output

#	Time	Action
1	19:35:08	Create database employee_det

USING A CREATED DATABASE

- **SYNTAX:**

  Use database_name;

- **Query:**

```
use employee_det;
```

- **Output:**

✓	2	19:43:02	use employee_det	0 row(s) affected	0.000 sec

CREATION OF TABLE 1

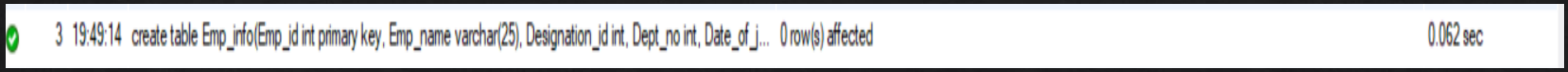
- **SYNTAX:**

  Create table table_name(“Columns to be added”);

- **Query:**

```
create table Emp_info(Emp_id int primary key, Emp_name varchar(25), Designation_id int, Dept_no int, Date_of_join date);
```

- **Output:**



DESCRIBE THE TABLE 1

- **SYNTAX:**

  describe table_name;

- **Query:**

```
describe Emp_info;
```

- **Output:**

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

Field	Type	Null	Key	Default	Extra
Emp_id	int	NO	PRI	NULL	
Emp_name	varchar(25)	YES		NULL	
Designation_id	int	YES		NULL	
Dept_no	int	YES		NULL	
Date_of_join	date	YES		NULL	

SHOW TABLE 1 IN DATABASE

- **SYNTAX:**

  show tables;

- **QUERY:**

```
show tables;
```

- **Output:**

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

	Tables_in_employee_det
▶	emp_info

INSERTING VALUES IN TABLE 1

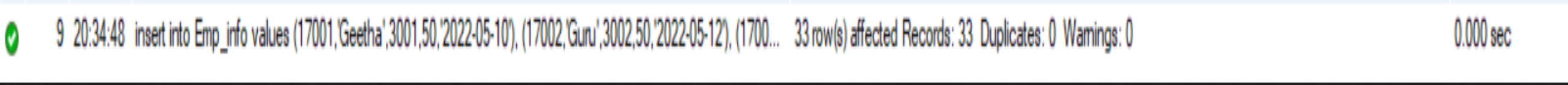
- **SYNTAX:**

insert into table_name value (value_column_name1, value_column_name2, value_column_name3, value_column_name4) ;

- **Query:**

```
insert into Emp_info values
(17001, 'Geetha',   3001,   50, '2022-05-10'),
(17002, 'Guru', 3002,   50, '2022-05-12'),
(17003, 'Gokul',    3003,   50, '2022-05-15'),
```

- **Output:**



DISPLAY INSERTED VALUES

- **SYNTAX:**

  Select * from table_name;

- **Query:**

```
select * from Emp_info;
```

- **Output:**

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Emp_id	Emp_name	Designation_id	Dept_no	Date_of_join
17001	Geetha	3001	50	2022-05-10
17002	Guru	3002	50	2022-05-12
17003	Gokul	3003	50	2022-05-15
17004	Mani	3004	60	2022-05-20
17005	Moorthy	3005	50	2022-05-23
17006	Amutha	3006	50	2022-06-05

CREATION AND INSERTION OF TABLE 2

- **Creating Query:**

```
create table salary_det(Salary_id int primary key, Emp_id int, salary_date date, branch_id int, amount int);
```

- **Creation Output:**

```
11  20:54:37  create table salary_det(Salary_id int primary key, Emp_id int, salary_date date, branch_id int, amount int)  0 row(s) affected
```

- **Inserting Query:**

```
insert into salary_det values
```

- **Insertion Output:**

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Salary_id	Emp_id	salary_date	branch_id	amount
18001	17001	2022-06-10	241	35000
18002	17002	2022-06-12	241	14000
18003	17003	2022-06-15	241	28000
18004	17004	2022-06-20	242	18000
18005	17005	2022-06-23	241	30000
18006	17006	2022-07-06	241	23000

salary_det 6

CREATION AND INSERTION OF TABLE 3

- **Creating Query:**

```
create table designation_det(Designation_id int primary key, Designation varchar(50));
```

- **Creation Output:**

14	22:18:14	create table designation_det(Designation_id int primary key, Designation varchar(50))	0 row(s) affected

- **Inserting Query:**

```
insert into designation_det values
(3001,  'Manager'),
(3002,  'Junior Associates'),
(3003,  'Senior Manager'),
```

- **Inserting Output:**

Result Grid | Filter Rows:

Designation_id	Designation
3001	Manager
3002	Junior Associates
3003	Senior Manager
3004	HR
3005	General Manager

CREATION AND INSERTION OF TABLE 4

- **Creating Query:**

```
create table department_det(dep_no int, dept_name varchar(40), branch_id int primary key, branch_name varchar(45));
```

- **Creation Output:**

```
17  22:33:33  create table department_det(dep_no int, dept_name varchar(40), branch_id int primary key, branch_name varc...  0 row(s) affected
```

- **Inserting Query:**

```
insert into department_det values
(50,    'Production Department',    241,    'Annan Nagar'),
(60,    'HR Department',    242,    'Velachery'),
(70,    'Sales Department', 243,    'Guindy'),
```

- **Inserting Output:**

Result Grid | Filter Rows: | Edit:

dep_no	dept_name	branch_id	branch_name
50	Production Department	241	Annan Nagar
60	HR Department	242	Velachery
70	Sales Department	243	Guindy
80	Finance Department	244	KMC
NULL	NULL	NULL	NULL

General Functions

- Where
- Or
- And
- In
- Not in
- >
- <
- >=
- <=
- < > (Not equal to)
- !

- Count
- Distinct
- Count With Distinct
- Order by Asc
- Order By Desc
- Group By
- Limit
- Desc Limit
- Like (_%)
- Not like
- Between

WHERE

This command is used to find the details of a specified value of a column.

Query:

```
select * from empl_details where Designation_ID=3001;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
17011	Manasi	3001	70	2022-06-10
17026	Keerthi	3001	60	2022-06-25
17033	Praveen	3001	80	2022-07-02

OR

This Command is used to find the details when any one condition is true among the multiple conditions.

Query:

```
select * from salary_details where Branch_ID=241 or Branch_ID=242;
```

Output:

Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
18001	17001	2022-06-10	241	35000
18002	17002	2022-06-12	241	14000
18003	17003	2022-06-15	241	28000
18004	17004	2022-06-20	242	18000
18005	17005	2022-06-23	241	30000
18006	17006	2022-07-06	241	23000
18008	17008	2022-07-08	242	18000
18009	17009	2022-07-09	241	30000

AND

This Command is used to find the details when all the conditions are true.

Query:

```
select * from empl_details where Dep_No=80 and Designation_ID=3002;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
17021	Veeramani	3002	80	2022-06-20
17022	Pandian	3002	80	2022-06-21
17023	Veera	3002	80	2022-06-22

IN

This command is used to find the details when all the conditions are true and is a easy way.

Query:

```
select * from empl_details where Designation_ID in (3005,3002);
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
17005	Moorthy	3005	50	2022-05-23
17009	Arthi	3005	50	2022-06-08
17012	Suja	3002	50	2022-06-11
17015	Sindhu	3005	80	2022-06-14
17016	Madhavi	3002	50	2022-06-15
17017	Swetha	3002	70	2022-06-16
17018	Selvi	3002	70	2022-06-17
17019	Pooja	3002	70	2022-06-18

NOT IN

This command will display the values except the values we give in the condition.

Query:

```
select * from empl_details where Designation_ID not in (3003,3008);
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
17004	Mani	3004	60	2022-05-20
17005	Moorthy	3005	50	2022-05-23
17006	Amutha	3006	50	2022-06-05
17008	Pavithra	3007	60	2022-06-07
17009	Arthi	3005	50	2022-06-08
17010	Kabilan	3006	70	2022-06-09
17011	Manasi	3001	70	2022-06-10
17012	Suja	3002	50	2022-06-11

GREATER THAN

Query:

```
select * from salary_details
where amount>30000;
```

Output:

Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
18001	17001	2022-06-10	241	35000
18011	17011	2022-07-11	243	35000
18026	17026	2022-07-26	242	35000
18033	17033	2022-08-02	244	35000

LESSER THAN

Query:

```
select * from salary_details
where amount<25000;
```

Output:

Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
18002	17002	2022-06-12	241	14000
18004	17004	2022-06-20	242	18000
18006	17006	2022-07-06	241	23000
18008	17008	2022-07-08	242	18000
18010	17010	2022-07-10	243	23000
18012	17012	2022-07-12	241	14000
18014	17014	2022-07-14	242	18000

GREATER THAN OR EQUAL TO

Query:

```
select * from salary_details
where Branch_ID >=244;
```

Output:

Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
18015	17015	2022-07-15	244	30000
18021	17021	2022-07-21	244	14000
18022	17022	2022-07-22	244	14000
18023	17023	2022-07-23	244	14000
18027	17027	2022-07-27	244	28000
18030	17030	2022-07-30	244	23000
18032	17032	2022-08-01	244	23000
18033	17033	2022-08-02	244	35000

LESSER THAN OR EQUAL TO

Query:

```
select * from salary_details
where Salary_ID <=18008;
```

Output:

Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
18001	17001	2022-06-10	241	35000
18002	17002	2022-06-12	241	14000
18003	17003	2022-06-15	241	28000
18004	17004	2022-06-20	242	18000
18005	17005	2022-06-23	241	30000
18006	17006	2022-07-06	241	23000
18007	17007	2022-07-07	243	28000
18008	17008	2022-07-08	242	18000

NOT EQUAL TO

Query:

```
select * from salary_details where Branch_ID <>244;
```

Output:

Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
18001	17001	2022-06-10	241	35000
18002	17002	2022-06-12	241	14000
18003	17003	2022-06-15	241	28000
18004	17004	2022-06-20	242	18000
18005	17005	2022-06-23	241	30000
18006	17006	2022-07-06	241	23000
18007	17007	2022-07-07	243	28000
18008	17008	2022-07-08	242	18000
18009	17009	2022-07-09	241	30000
18010	17010	2022-07-10	243	23000
18011	17011	2022-07-11	243	35000
18012	17012	2022-07-12	241	14000

COUNT

Query:

```
select count(Emp_name) from
empl_details;
```

Output:

	count(Emp_name)
▶	31

DISTINCT

Query:

```
select distinct Branch_ID from
salary_details;
```

Output:

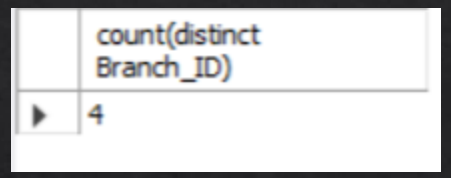
	Branch_ID
▶	241
	242
	243
	244

COUNT WITH DISTINCT

Query:

```
select count(distinct Branch_ID) from salary_details;
```

Output:



	count(distinct Branch_ID)
▶	4

ORDER BY ASC

Query:

```
select * from empl_details
order by Emp_name asc;
```

Output:

▶	17006	Amutha	3006	50	2022-06-05
	17009	Arthi	3005	50	2022-06-08
	17013	Arun	3003	60	2022-06-12
	17014	Deepa	3004	60	2022-06-13
	17025	Devan	3006	60	2022-06-24
	17024	Devi	3005	70	2022-06-23
	17032	ganesan	3006	80	2022-07-01
	17003	Gokul	3003	50	2022-05-15

ORDER BY DESC

Query:

```
select * from empl_details
order by Emp_name desc;
```

Output:

	Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
▶	17027	Venkatesh	3003	80	2022-06-26
	17021	Veeramani	3002	80	2022-06-20
	17023	Veera	3002	80	2022-06-22
	17017	Swetha	3002	70	2022-06-16
	17012	Suja	3002	50	2022-06-11
	17031	srinivasan	3005	70	2022-06-30
	17015	Sindhu	3005	80	2022-06-14
	17018	Selvi	3002	70	2022-06-17

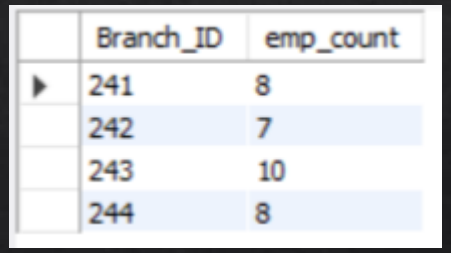
GROUP BY

This command is used to group the values based on the condition.

Query:

```
select Branch_ID, count(Emp_ID) as emp_count from salary_details group by Branch_ID;
```

Output:



Branch_ID	emp_count
241	8
242	7
243	10
244	8

LIMIT

This command is used to display the values to a limited level.

Query:

```
select * from salary_details limit 0,5;
```

Output:

Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
18001	17001	2022-06-10	241	35000
18002	17002	2022-06-12	241	14000
18003	17003	2022-06-15	241	28000
18004	17004	2022-06-20	242	18000
18005	17005	2022-06-23	241	30000

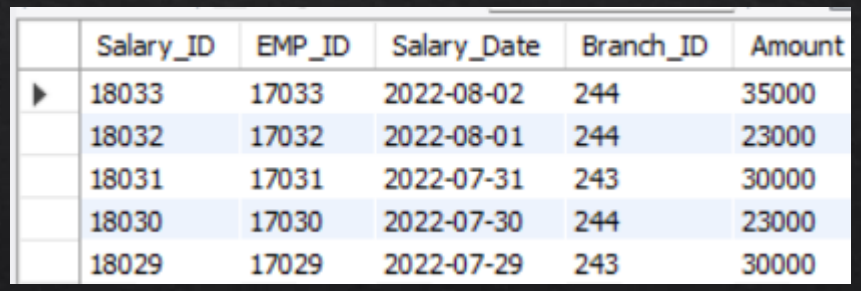
DESC LIMIT

This command is used to display the values in a descending order of limit mentioned.

Query:

```
select * from salary_details order by Emp_ID desc limit 0,5;
```

Output:



Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
18033	17033	2022-08-02	244	35000
18032	17032	2022-08-01	244	23000
18031	17031	2022-07-31	243	30000
18030	17030	2022-07-30	244	23000
18029	17029	2022-07-29	243	30000

LIKE

Query:

```
select * from empl_details
where Emp_name like 'Ja%';
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
17007	Jaga	3003	70	2022-06-06
NULL	NULL	NULL	NULL	NULL

NOT LIKE

Query:

```
select * from empl_details
where Emp_name not like
'Ja%';
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
17003	Gokul	3003	50	2022-05-15
17004	Mani	3004	60	2022-05-20
17005	Moorthy	3005	50	2022-05-23
17006	Amutha	3006	50	2022-06-05
17008	Pavithra	3007	60	2022-06-07
17009	Arthi	3005	50	2022-06-08

BETWEEN

This command is used to display the values between or in the range of two values.

Query:

```
select * from salary_details where amount between 10000 and 20000;
```

Output:

	Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
▶	18002	17002	2022-06-12	241	14000
	18004	17004	2022-06-20	242	18000
	18008	17008	2022-07-08	242	18000
	18012	17012	2022-07-12	241	14000
	18014	17014	2022-07-14	242	18000
	18016	17016	2022-07-16	241	14000
	18017	17017	2022-07-17	243	14000

MySQL String Functions

- Lcase
- Ucase
- Left
- Right
- Concat
- Trim
- Char_Length
- Mid
- Length

L CASE

This command is used to display all the characters in Lower case.

Query:

```
select*, lcase(Emp_name) as staff_name from empl_details;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	staff_name
17003	Gokul	3003	50	2022-05-15	gokul
17004	Mani	3004	60	2022-05-20	mani
17005	Moorthy	3005	50	2022-05-23	moorthy
17006	Amutha	3006	50	2022-06-05	amutha
17007	Jaga	3003	70	2022-06-06	jaga
17008	Pavithra	3007	60	2022-06-07	pavithra
17009	Arthi	3005	50	2022-06-08	arthi

U CASE

This command is used to display all the characters in Upper case.

Query:

```
select*, ucase(Emp_name) as STAFF_name from empl_details;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	STAFF_name
17003	Gokul	3003	50	2022-05-15	GOKUL
17004	Mani	3004	60	2022-05-20	MANI
17005	Moorthy	3005	50	2022-05-23	MOORTHY
17006	Amutha	3006	50	2022-06-05	AMUTHA
17007	Jaga	3003	70	2022-06-06	JAGA
17008	Pavithra	3007	60	2022-06-07	PAVITHRA
17009	Arthi	3005	50	2022-06-08	ARTHI
17010	Kabilan	3006	70	2022-06-09	KABILAN

RIGHT & LEFT

This command is used to display the specified characters from a data on either sides.

Query:

```
select left(Emp_ID,3) as left_id, right(Emp_ID,2) as right_id from empl_details;
```

Output:

left_id	right_id
170	03
170	04
170	05
170	06
170	07
170	08
170	09
170	10
170	11

CONCAT

This command is used to Concatenate two strings.

Query:

```
select concat(Emp_ID,"-",Emp_name) as naming_the_id from empl_details;
```

Output:

naming_the_id
17003-Gokul
17004-Mani
17005-Moorthy
17006-Amutha
17007-Jaga
17008-Pavithra
17009-Arthi
17010-Kabilan

TRIM

This command is used to trim the unwanted space and characters of the value.

Query:

```
select trim(Emp_name) from empl_details;
```

Output:

	trim(Emp_name)
▶	Gokul
	Mani
	Moorthy
	Amutha
	Jaga
	Pavithra
	Arthi
	Kabilan
	Manasi

MID

This Command is used to find the middle part of the characters in a string.

Query:

```
select Emp_ID, mid(Emp_ID,4,2) as Middle_id from empl_details;
```

Output:

Emp_ID	Middle_id
17003	03
17004	04
17005	05
17006	06
17007	07
17008	08
17009	09

LENGTH

This command is used to find given number of character in a string.

Query:

```
select * from empl_details where length(Emp_name)=5;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
17003	Gokul	3003	50	2022-05-15
17009	Arthi	3005	50	2022-06-08
17014	Deepa	3004	60	2022-06-13
17018	Selvi	3002	70	2022-06-17
17019	Pooja	3002	70	2022-06-18
17023	Veera	3002	80	2022-06-22

CHAR_LENGTH

This command is used to find the length of the characters of a string.

Query:

```
select Emp_ID,Emp_name, char_length(Emp_name) from empl_details;
```

Output:

Emp_ID	Emp_name	char_length(Emp_name)
17003	Gokul	5
17004	Mani	4
17005	Moorthy	7
17006	Amutha	6
17007	Jaga	4
17008	Pavithra	8
17009	Arthi	5
17010	Kabilan	7

MySQL Date Functions

- Date ADD
- Datediff
- Timestamp Diff
- Date Format
- Year
- Day
- Month
- Now

DATE DIFF

Query:

```
select*, datediff(curdate(), Date_of_Join)as EMP_exp from empl_details;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	EMP_exp
17003	Gokul	3003	50	2022-05-15	675
17004	Mani	3004	60	2022-05-20	670
17005	Moorthy	3005	50	2022-05-23	667
17006	Amutha	3006	50	2022-06-05	654
17007	Jaga	3003	70	2022-06-06	653

TIMESTAMP DIFF

Query:

```
select*, timestampdiff(year,Date_of_Join,curdate()) as EMP_exp from
empl_details;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	EMP_exp
17003	Gokul	3003	50	2022-05-15	1
17004	Mani	3004	60	2022-05-20	1
17005	Moorthy	3005	50	2022-05-23	1
17006	Amutha	3006	50	2022-06-05	1
17007	Jaga	3003	70	2022-06-06	1
17008	Pavithra	3007	60	2022-06-07	1
17009	Arthi	3005	50	2022-06-08	1

DATE FORMAT

Query:

```
select*, date_format (Date_of_Join,'%b') as month_name from empl_details
where date_format (Date_of_Join,'%b') like '_a%';
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	month_name
17003	Gokul	3003	50	2022-05-15	May
17004	Mani	3004	60	2022-05-20	May
17005	Moorthy	3005	50	2022-05-23	May

YEAR

Query:

```
select * from empl_details where year(Date_of_Join)=2022;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join
17003	Gokul	3003	50	2022-05-15
17004	Mani	3004	60	2022-05-20
17005	Moorthy	3005	50	2022-05-23
17006	Amutha	3006	50	2022-06-05
17007	Jaga	3003	70	2022-06-06
17008	Pavithra	3007	60	2022-06-07
17009	Arthi	3005	50	2022-06-08

JOIN QUERIES

- Inner Join
- Left Join
- Right Join
- Cross join
- Full Join

INNER JOIN

Query:

```
select * from empl_details inner join salary_details on empl_details.Emp_ID=
salary_details.Emp_ID;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
17003	Gokul	3003	50	2022-05-15	18003	17003	2022-06-15	241	28000
17004	Mani	3004	60	2022-05-20	18004	17004	2022-06-20	242	18000
17005	Moorthy	3005	50	2022-05-23	18005	17005	2022-06-23	241	30000
17006	Amutha	3006	50	2022-06-05	18006	17006	2022-07-06	241	23000
17007	Jaga	3003	70	2022-06-06	18007	17007	2022-07-07	243	28000
17008	Pavithra	3007	60	2022-06-07	18008	17008	2022-07-08	242	18000
17009	Arthi	3005	50	2022-06-08	18009	17009	2022-07-09	241	30000

LEFT JOIN

Query:

```
select * from empl_details left join Department_Det on empl_details.Dep_No=
Department_Det.Dep_No order by empl_details.Emp_ID;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	Dep_No	Dept_Name	Branch_ID	Branch_Name
17003	Gokul	3003	50	2022-05-15	50	Production Department	241	Annan Nagar
17004	Mani	3004	60	2022-05-20	60	HR Department	242	Velachery
17005	Moorthy	3005	50	2022-05-23	50	Production Department	241	Annan Nagar
17006	Amutha	3006	50	2022-06-05	50	Production Department	241	Annan Nagar
17007	Jaga	3003	70	2022-06-06	70	Sales Department	243	Guindy

RIGHT JOIN

Query:

```
select * from empl_details right join Designation_Det on empl_details.Designation_ID=
Designation_Det.Designation_ID;
```

Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	Designation_ID	Designation
17033	Praveen	3001	80	2022-07-02	3001	Manager
17026	Keerthi	3001	60	2022-06-25	3001	Manager
17011	Manasi	3001	70	2022-06-10	3001	Manager
17023	Veera	3002	80	2022-06-22	3002	Junior Associates
17022	Pandian	3002	80	2022-06-21	3002	Junior Associates

CROSS JOIN

Query:

```
select * from Designation_Det cross join Department_Det;
```

Output:

Designation_ID	Designation	Dep_No	Dept_Name	Branch_ID	Branch_Name
3001	Manager	80	Finance Department	244	KMC
3001	Manager	70	Sales Department	243	Guindy
3001	Manager	60	HR Department	242	Velachery
3001	Manager	50	Production Department	241	Annan Nagar
3002	Junior Associates	80	Finance Department	244	KMC
3002	Junior Associates	70	Sales Department	243	Guindy
3002	Junior Associates	60	HR Department	242	Velachery
3002	Junior Associates	50	Production Department	241	Annan Nagar
3003	Senior Manager	80	Finance Department	244	KMC

FULL JOIN

Query:

```
(select * from empl_details left join salary_details on
empl_details.Emp_ID=salary_details.Emp_ID) union(select * from empl_details right join
salary_details on empl_details.Emp_ID=salary_details.Emp_ID);
```

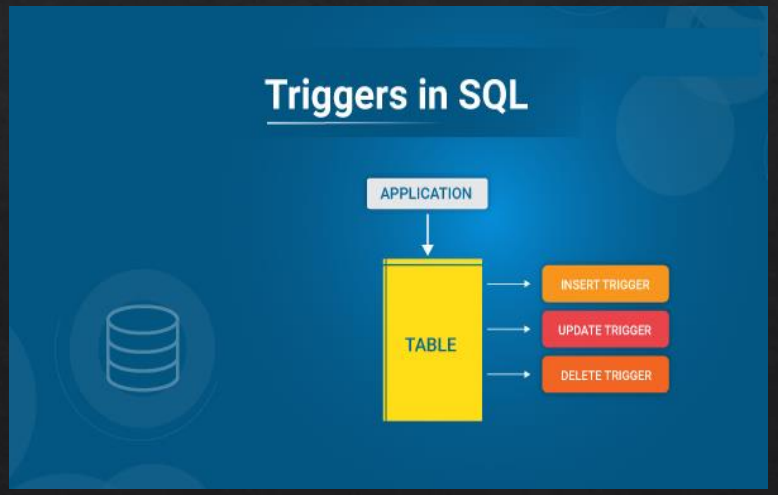
Output:

Emp_ID	Emp_name	Designation_ID	Dep_No	Date_of_Join	Salary_ID	EMP_ID	Salary_Date	Branch_ID	Amount
17003	Gokul	3003	50	2022-05-15	18003	17003	2022-06-15	241	28000
17004	Mani	3004	60	2022-05-20	18004	17004	2022-06-20	242	18000
17005	Moorthy	3005	50	2022-05-23	18005	17005	2022-06-23	241	30000
17006	Amutha	3006	50	2022-06-05	18006	17006	2022-07-06	241	23000
17007	Jaga	3003	70	2022-06-06	18007	17007	2022-07-07	243	28000

TRIGGERS

- Triggers are database operations which are automatically performed when an action such as Insert, Update or Delete is performed on a Table or a View in database.
- Triggers are associated with the Table or View directly i.e. each table has its own Triggers.

- Before Insert
- After Insert
- Before Update
- After Update
- Before Delete
- After Delete



TRIGGER BEFORE INSERT

- CREATE TABLE:

```
177  create table trigger_det (s_no int, column_no int, rows_afft int, primary key(s_no));
```

```
179  delimiter //
180  create trigger values_det before insert on trigger_det for each row
181  begin
182  if new.rows_afft < 5 then set new.rows_afft = 0;
183  end if ;
184  end //
185  delimiter ;
```

```
187  insert into trigger_det values
188  (1,1,2),
189  (2,2,3),
190  (3,3,6),
191  (4,4,5),
192  (5,5,7);
193
```

Result Grid | Filter Rows:

s_no	column_no	rows_afft
1	1	0
2	2	0
3	3	6
4	4	5
5	5	7
NULL	NULL	NULL

TRIGGER BEFORE UPDATE

BEFORE:

STUD_ID	STUD_NAME	CITY	ROLLNO	AGE	RESULT	MARKS	COMMON
1	Vasanth	Erode	143001	20	No Rank	37	1
2	Guru	Tiruppur	143002	20	No Rank	28	2
3	Gokul	Tiruchirapalli	143003	20	Average	28	3
4	Mani	Kumarapalayam	143004	24	No Rank	31	4
5	Moorthy	Salem	143005	18	Very Good	86	5
6	Amutha	Chennai	143006	20	Average	61	6
7	Jaga	Madurai	143007	24	Very Good	89	NULL
8	Pavithra	Erode	143008	20	Average	68	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
224  delimiter //
225  create trigger before_update before update on stud_det for each row
226  begin
227  if new.city='salem' then set
228  new.city='this is my native';
229  end if ;
230  end //
231  delimiter ;
232
233  update stud_det set city='salem' where stud_id=1;
234  select * from stud_det;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

STUD_ID	STUD_NAME	CITY	ROLLNO	AGE	RESULT	MARKS	COMMON_ID
1	Vasanth	this is my native	143001	20	No Rank	37	1
2	Guru	Tiruppur	143002	20	No Rank	28	2
3	Gokul	Tiruchirapalli	143003	20	Average	28	3
4	Mani	Kumarapalayam	143004	24	No Rank	31	4
5	Moorthy	Salem	143005	18	Very Good	86	5
6	Amutha	Chennai	143006	20	Average	61	6
7	Jaga	Madurai	143007	24	Very Good	89	NULL
8	Pavithra	Erode	143008	20	Average	68	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

TRIGGER BEFORE DELETE

```
236  create table det_info_stud (stud_id int primary key, delete_dat timestamp default now());
237
238  delimiter //
239  create trigger before_delete before delete on stud_det for each row
240  begin
241  insert into det_info_stud(stud_id)
242  values (old.stud_id);
243  end //
244  delimiter ;
245
246  delete from stud_det where STUD_ID=2;
247  select * from det_info_stud;
248
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

stud_id	delete_dat
2	2024-03-05 01:40:06
NULL	NULL