MY SQL PROJECT

BY

SESHA KARTHICK.G


MY SQL

SQL

- SQL is a standard language for storing, manipulating and retrieving data in databases.
- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

WHAT CAN SQL 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

DBMS VS RDBMS

DBMS

- DBMS stands for Database Management System.
- DBMS stores data as file .
- In DBMS data elements need to access individually.
- It deals with small quantity of data.
- It Supports single user.

RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS stores data in tabular form.
- In RDBMS multiple data elements can be accessed at the same time
- It deals with large amount of data.
- It supports multiple users.

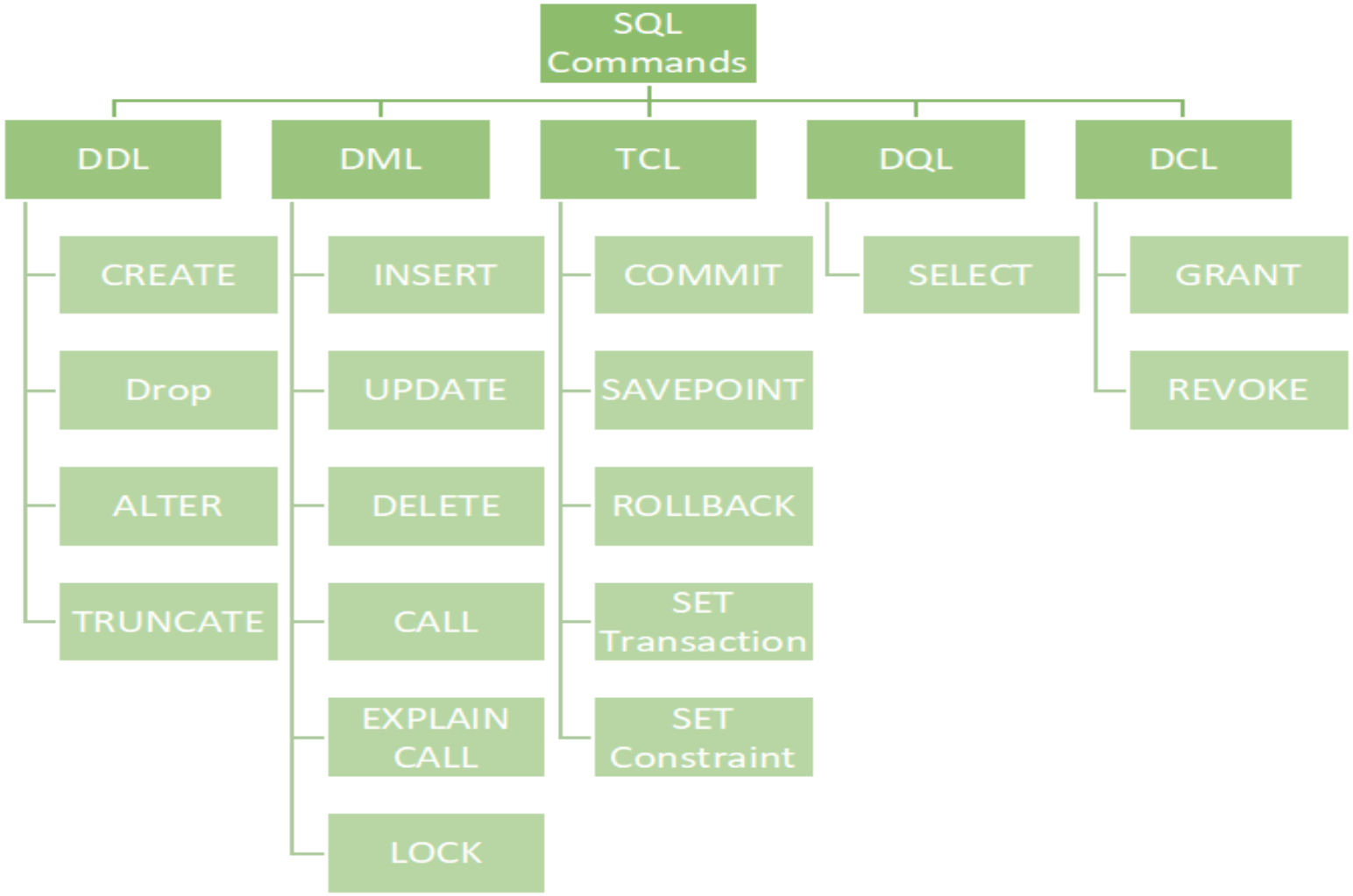
SQL COMMANDS

SQL COMMANDS

These SQL commands are categorized into Several main groups:

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

SQL COMMANDS



DATA TYPES

DATA TYPES IN MYSQL

- Data types define the kind of information stored.
- **Numeric Data Type** -Stores numbers, like ages or prices.
- **Date Time Data Type** - Holds dates and times, such as birthdays.
- **String Data Type** - Contains text, like names and addresses.

Datatype	Description
INT(size)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)
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
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
FLOAT(p)	A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()

CONSTRAINTS

CONSTRAINTS IN MYSQL

- PRIMARY KEY - Uniquely identifies each row, ensuring no duplicates.
- FOREIGN KEY - Establishes relationships between tables, enforcing referential integrity.
- UNIQUE - Ensures that values in a column are unique across all rows.
- CHECK - Verifies that values in a column meet specified conditions.
- NOT NULL - Requires a column to have a value, preventing NULL entries.
- DEFAULT - Sets a default value for a column when no value is provided.
- INDEX - Improves data retrieval speed by creating an index for the column.
- AUTO_INCREMENT - Automatically assigns a unique value, typically used with primary keys.

MAIN COMMANDS

MAIN COMMANDS IN MYSQL:

- Main commands are like actions you can perform.
- CREATE - Used to make new database objects like tables.
- USE - Specifies which database to work within.
- SHOW - Displays database information like tables.
- RENAME - Changes the name of an existing table.
- ALTER - Modifies the structure of a table.
- DESCRIBE - Shows details about a table's structure.
- UPDATE - Modifies existing data in a table.
- DELETE - Removes data from a table.
- DROP - Deletes an entire table or database.

CREATING A DATABASE .

- **1, Creating a Database**

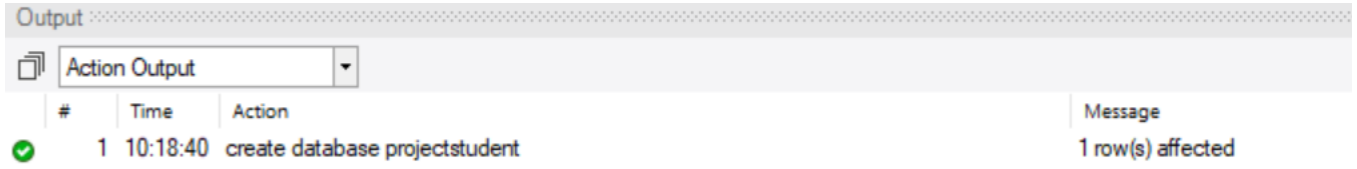
  Syntax :

  Create Database database_name;

  Query:

  create database projectstudent;

  Output :



  After running the above command successfully, you should see a confirmation message indicating that the database has been created.

USING A DATABASE

- **2. Using a Database**

  Syntax:

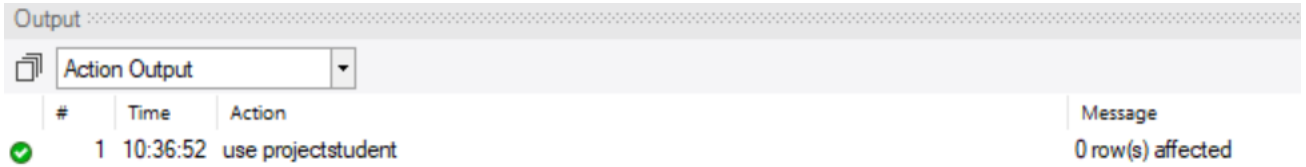
  Use database_name;

  Query:

  Using the “projectstudent” Database :

  use projectstudent;

  Output :



  After executing the “Use” command, you will switch to the specified database, in this case, the “projectstudent” database.

SHOW DATABASES

3. Show Databases

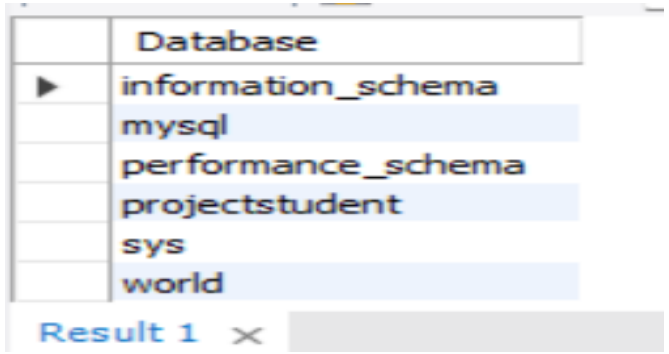
Syntax :

Show Databases;

Query :

Show databases;

Output :



When you execute the “Show databases" command, you will receive a result that displays a list of all databases available on the MySQL server. Each database name will be listed in a separate row.

DROP DATABASE :

4. Drop Database

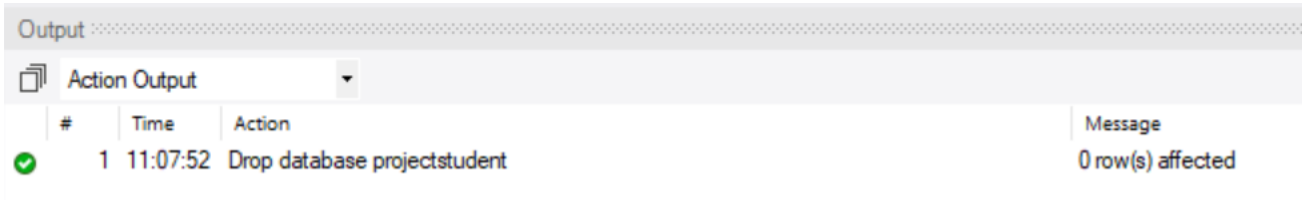
syntax:

Drop database database_name;

Query:

Drop database projectstudent;

Output:



After executing the " Drop database projectstudent" query , The projectstudent database is deleted

TABELS IN PROJECT_STUDENT DATABASE :

- Table 1 (STUDENT_INFORMATION)
- Table 2 (COURSE_INFORMATION)
- Table 3 (MARKS_INFO)

4. Table creation

Syntax :

Create table table_name (column1 datatype, column2 datatype, ... columnN datatype);

Insert into table_name (column1, column2, column3, ...)Values (value1, value2, value3, ...);

Query :

Table 1 (Student_information)

Table Creation Statement for stud_info:

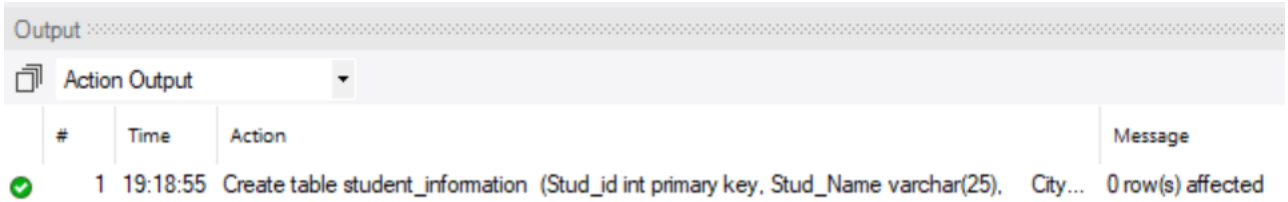
Create table student_information

(Stud_id int primary key, Stud_Name varchar(25),

City_State varchar(25), Age int, Roll_no int,

Community varchar(10), Course_ID int);

Output:



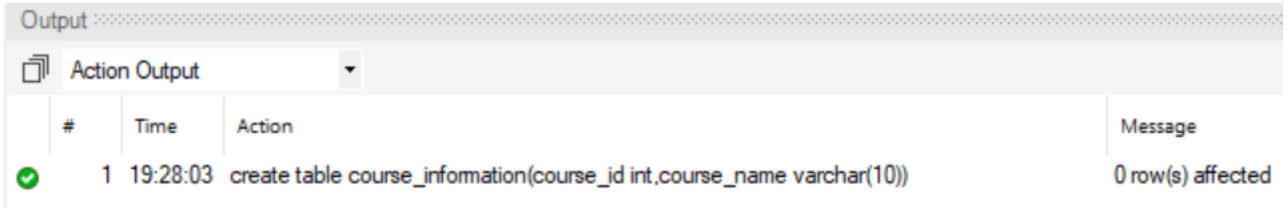
- **Table 2 (Course_information)**

  This Course_information table contains data of Course_id , Course_name.

  Query :

  create table course_information(course_id int,course_name varchar(10));

  Output:



- **Table 3 (marks_information)**

  This marks_information table contains data of student's marks_id , Student_id , finance , cost , corporate .

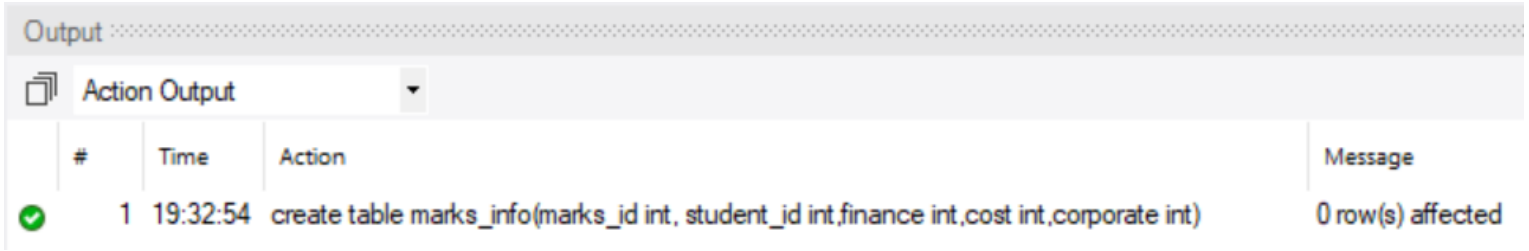
  Query:

  create table marks_info(marks_id int,

  student_id int,finance int,

  cost int,corporate int);

  Output :



SHOW TABLES

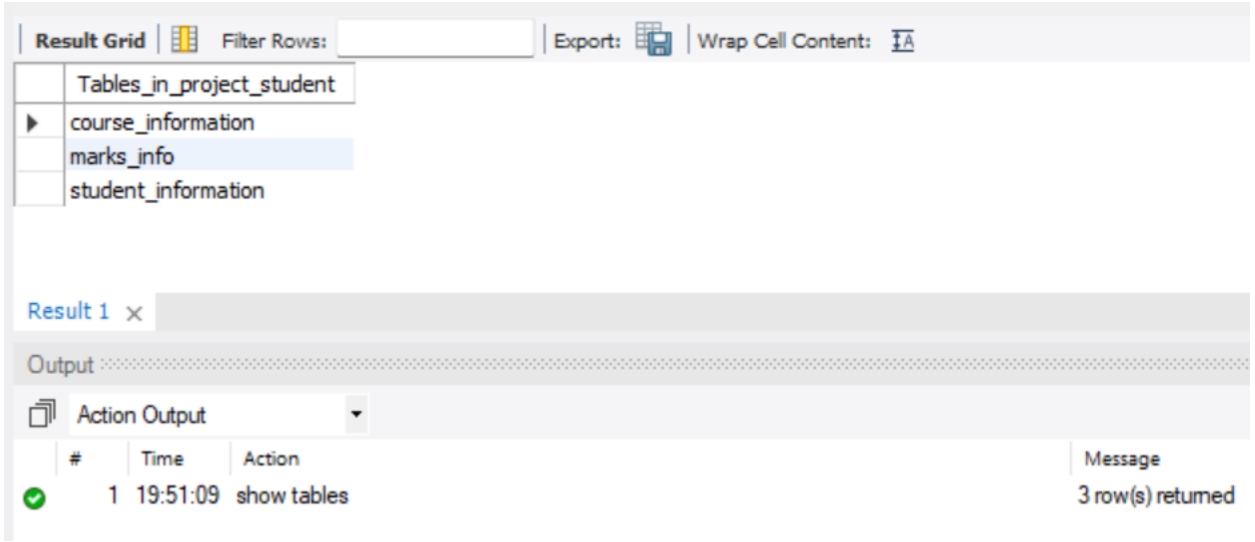
- 5, SHOW TABLES

  Show all tables in project_student:

  Query:

  show tables;

  Output:



INSERTING VALUES INTO TABLES

- **INSERT VALUES INTO TABLE 1 (Student_information)**

Syntax:

INSERT INTO table_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

(OR)

INSERT INTO table_name

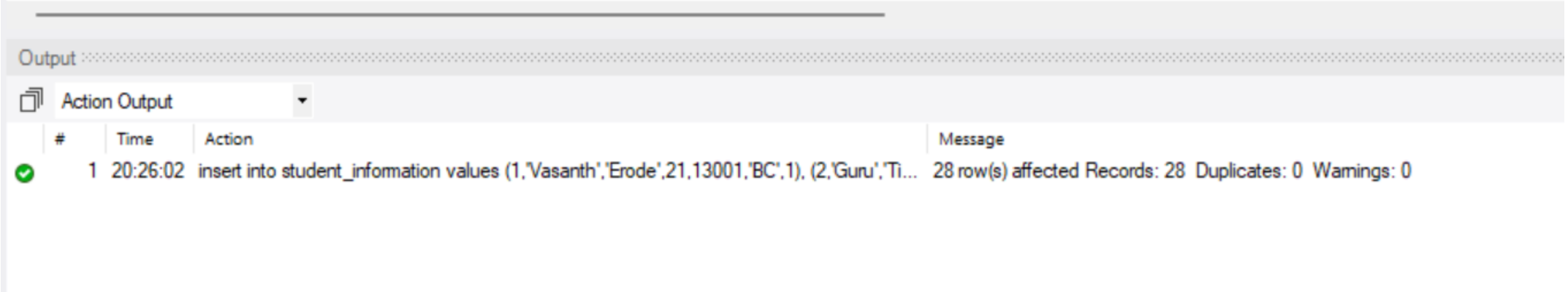
VALUES (value1, value2, value3, ...);

- **Query :**

```
insert into student_information values
(1,  'Vasanth',  'Erode',  21,  13001,  'BC',  1),
(2,  'Guru',  'Tiruppur',  20,  13002,  'MBC',  1),
(3,  'Gokul',  'Tiruchirapalli',  18,  13003,  'SC',  1),
(4,  'Mani',  'Kumarapalayam',24,  13004,  'BC',  1),
(5,  'Moorthy',  'Salem',  18,  13005,  'MBC',  2),
(6,  'Amutha',  'Chennai',  17,  13006,  'SC',  2),
(7,  'Jaga',  'Madurai',  24,  13007,  'BC',  2),
(8,  'Pavithra',  'Erode',  23,  13008,  'MBC',  2),
(9,  'Arthi',  'Tiruppur',  17,  13009,  'SC',  3),
(10,  'Kabilan',  'Tiruchirapalli',  24,  13010,  'BC',  3),
(11,  'Manasi',  'Kumarapalayam',17,  13011,  'MBC',  3),
(12,  'Suja','Salem',  23,  13012,  'SC',  3),
(13,  'Arun',  'Chennai',  22,  13013,  'BC',  3),
(14,  'Deepa',  'Madurai',  20,  13014,  'MBC',  1),
(15,  'Sindhu',  'Erode',  22,  13015,  'SC',  1),
```

```
(16, 'Madhavi',     'Tiruppur',      20, 13016, 'BC', 1),
(17, 'Swetha', 'Tiruchirapalli',     17, 13017, 'MBC',    1),
(18, 'Selvi',    'Kumarapalayam', 22, 13018, 'SC', 3),
(19, 'Pooja',   'Salem',   19, 13019,  'BC', 1),
(20, 'Lakshmi', 'Chennai',       17, 13020, 'MBC',    2),
(21, 'Veeramani',  'Madurai',     21, 13021, 'SC', 1),
(22, 'Pandian',      'Erode',   21, 13022, 'BC', 1),
(23, 'Veera',   'Tiruppur',      20, 13023, 'MBC',    2),
(24, 'Devi',     'Tiruchirapalli',     20, 13024, 'SC', 3),
(25, 'Devan',  'Kumarapalayam', 21, 13025, 'BC', 1),
(26, 'Keerthi', 'Salem',   17, 13026,  'MBC',    2),
(27, 'Venkatesh',    'Chennai',      24, 13027, 'SC', 1),
(28, 'Raja',     'Madurai',      24, 13028, 'SC', 3);
```

- **Output (inserted values into table 1 student_information) :**

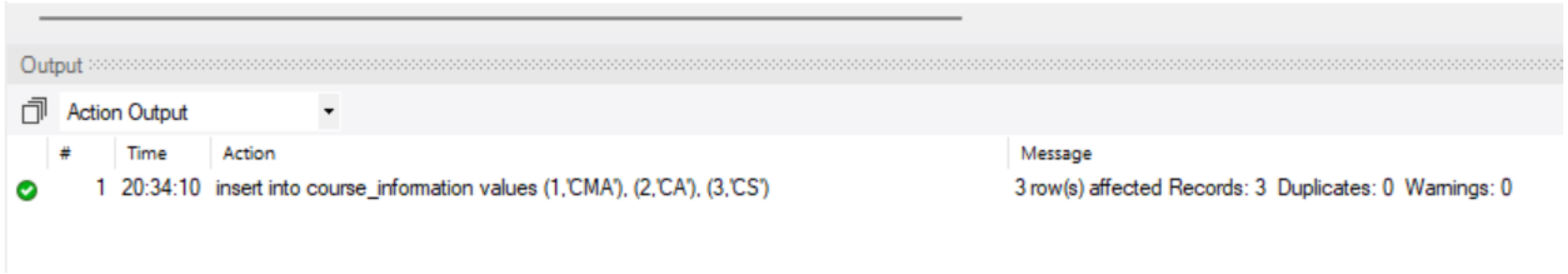


INSERT VALUES INTO TABLE 2 (COURSE_INFORMATION)

- **Query:**

```
insert into course_information values   (1,'CMA'),(2,'CA'),(3,'CS');
```

Output:



INSERT VALUES INTO TABLE 3 (MARKS_INFO)

- Query

```
insert into marks_info values
(14001,    1,    75,    76,    65),
(14002,    2,    92,    90,    19),
(14003,    3,    38,    37,    46),
(14004,    4,    39,    90,    58),
(14005,    5,    34,    89,    20),
(14006,    6,    44,    38,    60),
(14007,    7,    50,    26,    98),
(14008,    8,    59,    78,    82),
(14009,    9,    89,    47,    88),
(14010,    10,   20,    25,    100),
(14011,    11,   74,    50,    100),
(14012,    12,   81,    62,    31),
(14013,    13,   60,    19,    33),
(14014,    14,   77,    22,    23),
(14015,    15,   68,    38,    35),
(14016,    16,   31,    60,    83),
```

(14017, 17, 83, 53, 79),

(14018, 18, 37, 79, 88),

(14019, 19, 55, 76, 76),

(14020, 20, 25, 40, 81),

(14021, 21, 38, 87, 48),

(14022, 22, 84, 63, 72),

(14023, 23, 80, 44, 64),

(14024, 24, 53, 46, 59),

(14025, 25, 79, 48, 94),

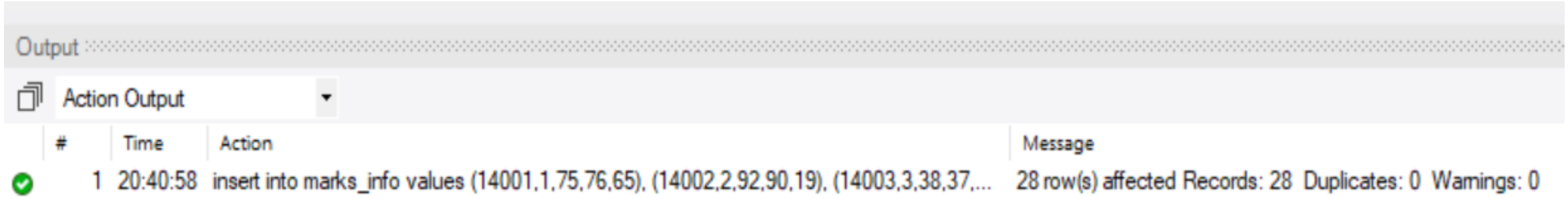
(14026, 26, 48, 66, 70),

(14027, 27, 61, 45, 74),

(14028, 28, 24, 95, 58);

- Output :

  Output for inserting values into table3 Marks_info



DROP OR REMOVE A TABLE

- Remove a table from database

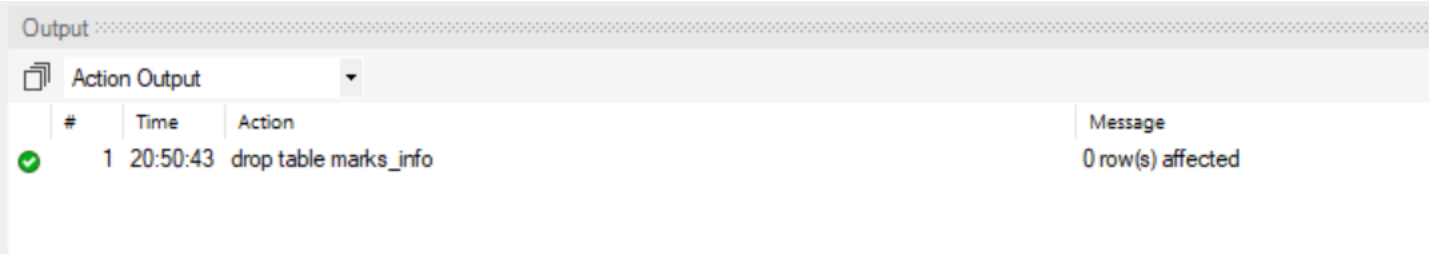
  Syntax:

  Drop table Table_name;

  Query :

  Drop table marks_info;

  Output:



MYSQL ALTER TABLE STATEMENT

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

ALTER TABLE - ADD Column

To add a column in a table.

Syntax:

ALTER TABLE table_name ADD column_name datatype;

Query:

alter table course_information add course_details varchar(30);

Output:

ALTER TABLE - MODIFY COLUMN

- To change the data type of a column in a table, use the following syntax:

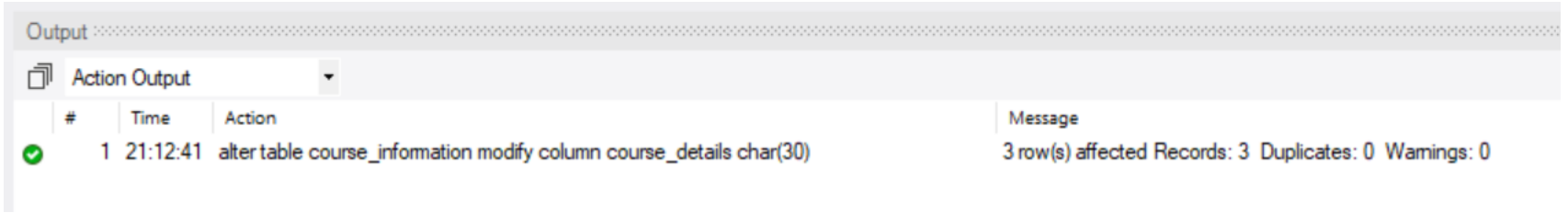
  Syntax:

  ALTER TABLE table_name MODIFY COLUMN column_name datatype;

  Query:

  alter table course_information modify column course_details char(30);

  Output:



ALTER TABLE - DROP COLUMN

- To delete a column in a table, use the following syntax

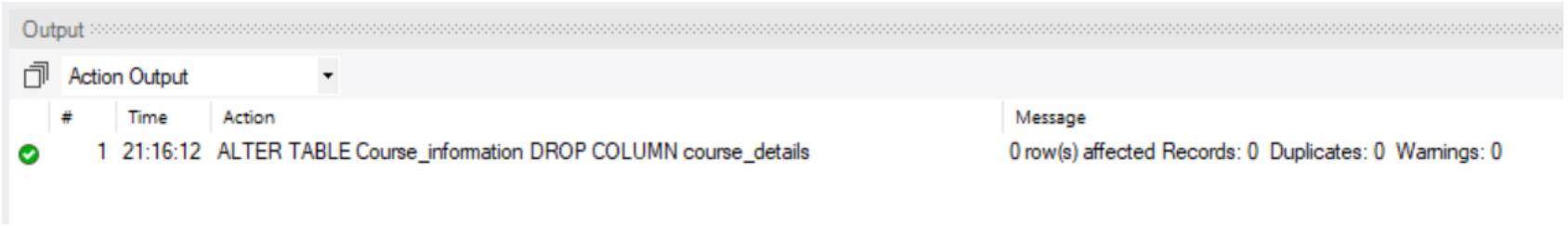
Syntax:

ALTER TABLE table_name DROP COLUMN column_name;

Query:

ALTER TABLE Course_information DROP COLUMN course_details;

Output:



ALTER TABLE –NOT NULL CONSTRAINT

- The NOT NULL constraint enforces a column to NOT accept NULL values.

  Add a not null constraint in student_information using alter command

  Syntax:

  Alter table table_name modify column_name Datatype NOT NULL;

  Query:

  alter table student_information modify stud_name varchar(25) not null;

  Output:

ALTER TABLE MODIFY DATA TYPE

- Alter Table Modify Data Type.

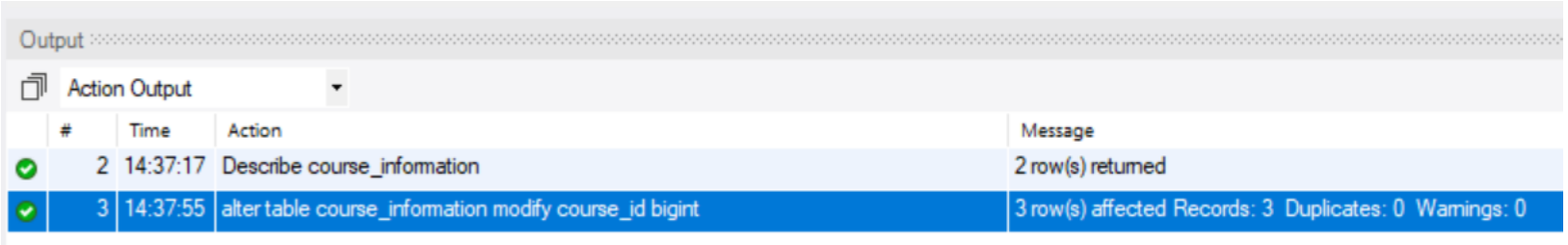
  Syntax:

  Alter table table_name modify column_name new data type;

  Query :

  Alter table course_information modify course_id bigint;

  Output:



  After executing this command there is no specific output message, but the data type of the "course_id" column in the "course_information" table is changed to bigint.

MYSQL UNIQUE CONSTRAINT

- The UNIQUE constraint ensures that all values in a column are different.

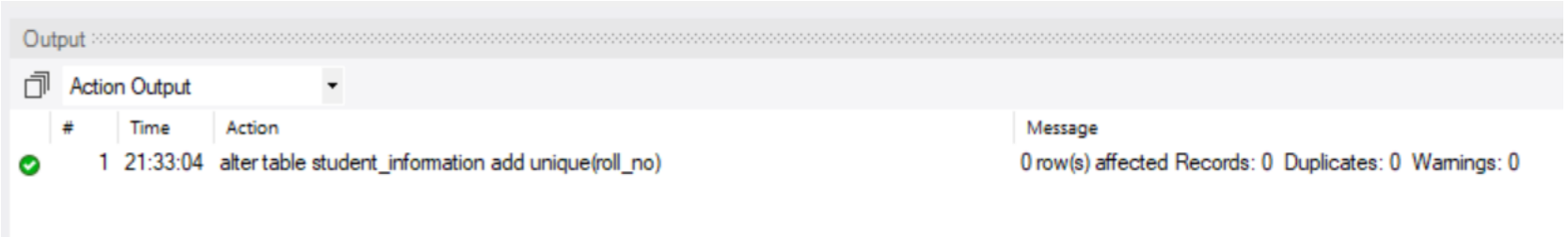
Syntax:

Alter table Table_name add unique (Column_name);

Query:

alter table student_information add unique (roll_no);

Output:



MYSQL PRIMARY KEY CONSTRAINT

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.

PRIMARY KEY on ALTER TABLE

Syntax :

Alter table table_name add primary key (column_name);

Query:

alter table student_information add primary key (stud_id);

Output:

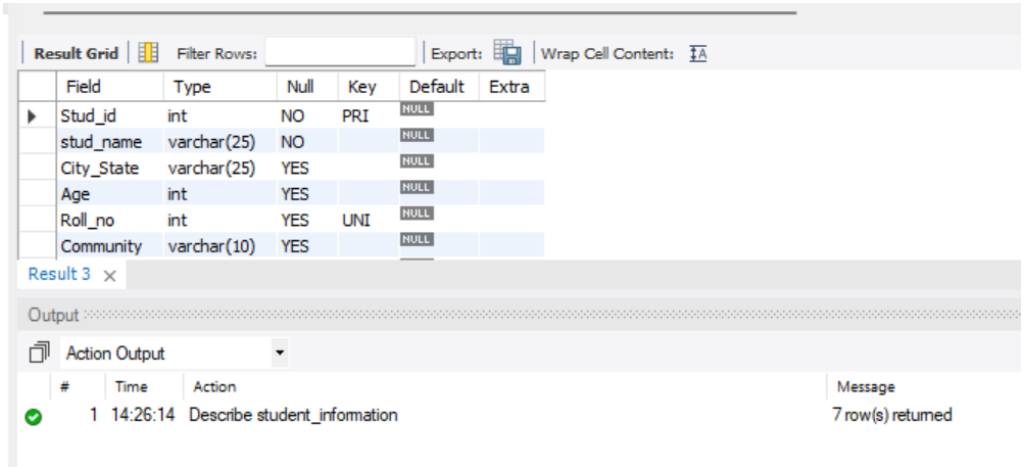
MYSQL DESCRIBE TABLE

- Describe Table:

  Syntax:

  Describe Table_name (or) Desc table_name;

  Query:



The DESCRIBE command provides information about the structure of the table, including the field names, data types, whether they can be NULL, any keys, and default values.

UPDATE TABLE TO CHANGE VALUES

- Update Table To Change Values

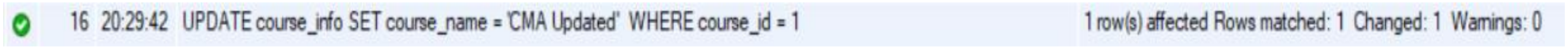
  Syntax:

  Update table_name Set Column_name = new_value where condition;

  Query:

  Update course_info Set course_name = 'CMA Updated' Where course_id = 1;

  Output:



  After executing this command there is no specific output message, but the course name for the course with `course_id` 1 is updated to 'CMA Updated'.

DELETE STATEMENTS

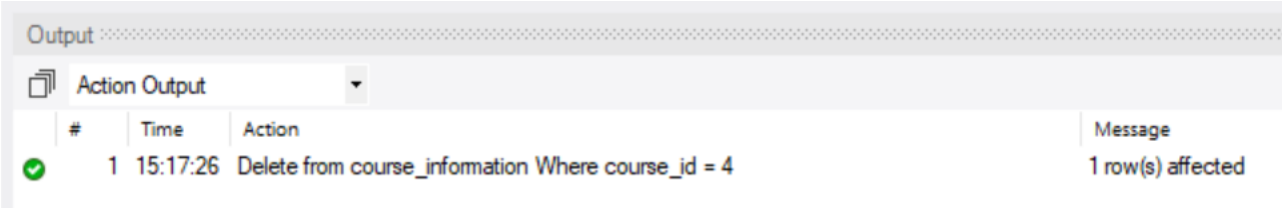
- Syntax :

  Delete from table_name Where condition;

  Query :

  Delete from course_information Where course_id = 4;

  Output:



  After executing this command there is no specific output message, but the course with `course_id` 4 is deleted from the "course_information" table.

GENERAL FUNCTIONS

GENERAL FUNCTIONS IN MYSQL

General Functions help filter and manipulate data.

1. WHERE - Like a filter to select specific data.
2. AND - Combine conditions.
3. BETWEEN - Checks if a value is within a range.
4. OR - Selects data if any condition is true.
5. IN - Checks if a value matches any in a list.
6. NOT IN - Checks if a value doesn't match any in a list.
7. Comparison Operators - >, <, = for comparing values.
8. COUNT and DISTINCT - Count rows and remove duplicates.
9. ORDER BY - Arrange data in ascending (ASC) or descending (DESC) order.
10. GROUP BY - Group data based on a common column.
11. LIMIT and OFFSET - Limit the number of rows and skip a certain number.
12. LIKE and NOT LIKE - Search for patterns in text.

- 1, WHERE

  Retrieve Students with an age greater than 23:

  Query:

  Select * from student_information Where Age > 23;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
4	Mani	Kumarapalayam	24	13004	BC	1
7	Jaga	Madurai	24	13007	BC	2
10	Kabilan	Tiruchirapalli	24	13010	BC	3
27	Venkatesh	Chennai	24	13027	SC	1
28	Raja	Madurai	24	13028	SC	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- **2. AND**

  Retrieve students who belong to the ‘MBC’ community and are enrolled in Course_ID 1:

  Select * from student_information Where Community = 'MBC' AND Course_ID = 1;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
2	Guru	Tiruppur	20	13002	MBC	1
14	Deepa	Madurai	20	13014	MBC	1
17	Swetha	Tiruchirapalli	17	13017	MBC	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- 3. BETWEEN

  Retrieve students with a Age between 13010 and 13012:

  Select * from student_information Where Roll_no BETWEEN 13010 AND 13012;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
10	Kabilan	Tiruchirapalli	24	13010	BC	3
11	Manasi	Kumarapalayam	17	13011	MBC	3
12	Suja	Salem	23	13012	SC	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- 4. OR

  Retrieve students from the ‘Erode’ city or ‘Chennai’ city:

  Select * from student_information Where City_State = 'Erode' OR City_State = 'Chennai’;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
1	Vasanth	Erode	21	13001	BC	1
6	Amutha	Chennai	17	13006	SC	2
8	Pavithra	Erode	23	13008	MBC	2
13	Arun	Chennai	22	13013	BC	3
15	Sindhu	Erode	22	13015	SC	1
20	Lakshmi	Chennai	17	13020	MBC	2
22	Pandian	Erode	21	13022	BC	1
27	Venkatesh	Chennai	24	13027	SC	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- **5. IN**
- **Retrieve students from the 'BC' and 'MBC' communities:**
- Select * from student_information Where Community IN ('BC', 'MBC');

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
1	Vasanth	Erode	21	13001	BC	1
2	Guru	Tiruppur	20	13002	MBC	1
4	Mani	Kumarapalayam	24	13004	BC	1
5	Moorthy	Salem	18	13005	MBC	2
7	Jaga	Madurai	24	13007	BC	2
8	Pavithra	Erode	23	13008	MBC	2
10	Kabilan	Tiruchirapalli	24	13010	BC	3
11	Manasi	Kumarapalayam	17	13011	MBC	3
13	Arun	Chennai	22	13013	BC	3
14	Deepa	Madurai	20	13014	MBC	1
16	Madhavi	Tiruppur	20	13016	BC	1
17	Swetha	Tiruchirapalli	17	13017	MBC	1
19	Pooja	Salem	19	13019	BC	1
20	Lakshmi	Chennai	17	13020	MBC	2
22	Pandian	Erode	21	13022	BC	1
23	Veera	Tiruppur	20	13023	MBC	2
25	Devan	Kumarapalayam	21	13025	BC	1
26	Keerthi	Salem	17	13026	MBC	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- **6. NOT IN**
- **Retrieve students who city/state are not in Chennai, Tiruppur, Kumarapalayam, Madurai, Erode, Tiruchirapalli** :
- Select * from student_information Where city_state not in ('chennai','tiruppur','kumarapalayam','madurai','erode','tiruchirapalli');

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
5	Moorthy	Salem	18	13005	MBC	2
12	Suja	Salem	23	13012	SC	3
19	Pooja	Salem	19	13019	BC	1
26	Keerthi	Salem	17	13026	MBC	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- 7. Comparison Operators (> , < , >= , <= , < > , !=)

  Retrieve students older than or equal to 23:

  Select * from student_information Where Age >= 23;

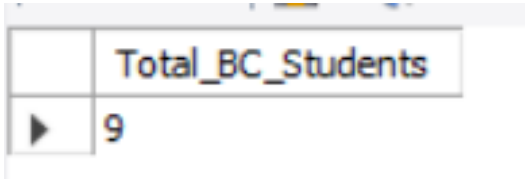
Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
4	Mani	Kumarapalayam	24	13004	BC	1
7	Jaga	Madurai	24	13007	BC	2
8	Pavithra	Erode	23	13008	MBC	2
10	Kabilan	Tiruchirapalli	24	13010	BC	3
12	Suja	Salem	23	13012	SC	3
27	Venkatesh	Chennai	24	13027	SC	1
28	Raja	Madurai	24	13028	SC	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Retrieve students with Roll_no less than 13004:

Select * from student_information Where Roll_no < 13004;

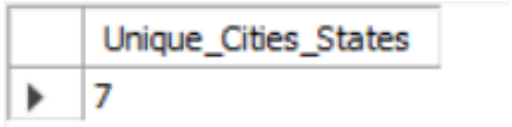
Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
1	Vasanth	Erode	21	13001	BC	1
2	Guru	Tiruppur	20	13002	MBC	1
3	Gokul	Tiruchirapalli	18	13003	SC	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- **9. COUNT**
- **Count the number of students in the 'BC' community:**
- Select Count(*) as Total_BC_Students from student_information WHERE Community = 'BC';



	Total_BC_Students
▶	9

- **10. DISTINCT**
- **Count the distinct cities/states among students:**
- Select count(Distinct City_State) as Unique_Cities_States from student_information;



	Unique_Cities_States
▶	7

- 11. ORDER BY

Retrieve students in ascending order of Age:

Select * from student_information Order by Age ASC;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
6	Amutha	Chennai	17	13006	SC	2
9	Arthi	Tiruppur	17	13009	SC	3
11	Manasi	Kumarapalayam	17	13011	MBC	3
17	Swetha	Tiruchirapalli	17	13017	MBC	1
20	Lakshmi	Chennai	17	13020	MBC	2
26	Keerthi	Salem	17	13026	MBC	2
3	Gokul	Tiruchirapalli	18	13003	SC	1
5	Moorthy	Salem	18	13005	MBC	2
19	Pooja	Salem	19	13019	BC	1
2	Guru	Tiruppur	20	13002	MBC	1
14	Deepa	Madurai	20	13014	MBC	1
16	Madhavi	Tiruppur	20	13016	BC	1
23	Veera	Tiruppur	20	13023	MBC	2

stud_info 42 ×

Retrieve students in descending order of Roll_no:

Select * from student_information order by Roll_no DESC;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
28	Raja	Madurai	24	13028	SC	3
27	Venkatesh	Chennai	24	13027	SC	1
26	Keerthi	Salem	17	13026	MBC	2
25	Devan	Kumarapalayam	21	13025	BC	1
24	Devi	Tiruchirapalli	20	13024	SC	3
23	Veera	Tiruppur	20	13023	MBC	2
22	Pandian	Erode	21	13022	BC	1

- **12. GROUP BY**

  Count the number of students in each community:

  Select Community, Count(*) as Total_Students from student_information GROUP BY Community;

Community	Total_Students
BC	9
MBC	9
SC	10

- **13. LIMIT and OFFSET**

  Retrieve the first 3 students:

  Select * from student_information LIMIT 3;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
1	Vasanth	Erode	21	13001	BC	1
2	Guru	Tiruppur	20	13002	MBC	1
3	Gokul	Tiruchirapalli	18	13003	SC	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

  Retrieve students 6 to 10:

  Select * from student_information LIMIT 5 OFFSET 5;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
6	Amutha	Chennai	17	13006	SC	2
7	Jaga	Madurai	24	13007	BC	2
8	Pavithra	Erode	23	13008	MBC	2
9	Arthi	Tiruppur	17	13009	SC	3
10	Kabilan	Tiruchirapalli	24	13010	BC	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- 14. LIKE AND NOT LIKE

Retrieve students with names containing ‘eer’

Select * from student_information Where Stud_Name LIKE '%eer%’;

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
21	Veeramani	Madurai	21	13021	SC	1
23	Veera	Tiruppur	20	13023	MBC	2
26	Keerthi	Salem	17	13026	MBC	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Retrieve students with names not containing ‘a’:

Select * from student_information Where Stud_Name NOT LIKE '%a%';

Stud_id	Stud_Name	City_State	Age	Roll_no	Community	Course_ID
2	Guru	Tiruppur	20	13002	MBC	1
3	Gokul	Tiruchirapalli	18	13003	SC	1
5	Moorthy	Salem	18	13005	MBC	2
15	Sindhu	Erode	22	13015	SC	1
18	Selvi	Kumarapalayam	22	13018	SC	3
24	Devi	Tiruchirapalli	20	13024	SC	3
26	Keerthi	Salem	17	13026	MBC	2

CALCULATED FUNCTIONS

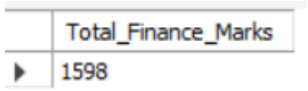
- **Calculated Functions in MySQL**
- Calculated Functions help with mathematical operations.

1. **SUM -** Adds values.
2. **AVG -** Calculates the average.
3. **ROUND** - Like a tool for rounding decimal numbers.
4. **MAX -** Finds the highest value.
5. **MIN -** Finds the lowest value.
6. **COUNT** - Counts data based on condition.

- 1. SUM

  Calculate the total Finance marks for all students:

  Select SUM(Finance) as Total_Finance_Marks from Marks_info;

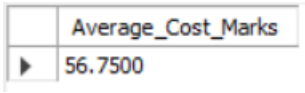


	Total_Finance_Marks
▶	1598

- 2. AVERAGE

  Calculate the average Cost marks for all students:

  Select Avg(Cost) as Average_Cost_Marks from marks_info;

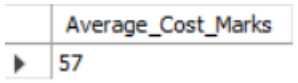


	Average_Cost_Marks
▶	56.7500

- **3. ROUND**

  Use the Round function in SQL to round the calculated average

  Select round(Avg(Cost),0) as Average_Cost_Marks from marks_info;



	Average_Cost_Marks
▶	57

- 4. MIN

  Find the minimum Corporate marks among all students:

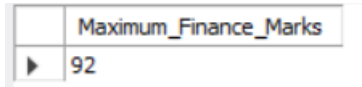
  Select MIN(Corporate) as Minimum_Corporate_Marks from Marks_info;

Minimum_Corporate_Marks
19

- **5. MAX**

  Find the maximum Finance marks among all students:

  Select MAX(Finance) as Maximum_Finance_Marks from Marks_info;

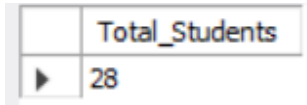


Maximum_Finance_Marks
92

- **6. COUNT**

  Count the number of rows (students) in the Marks_info table:

  Select COUNT(*) as Total_Students from Marks_info;



Total_Students
28

STRING FUNCTIONS

String Functions in MySQL

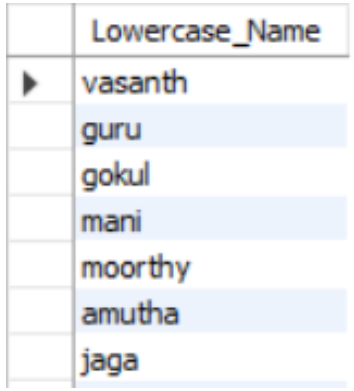
- String functions are like tools for working with text.

1. Lower - Changes all letters to lowercase in a text.
2. Upper - Converts all letters to uppercase in a text.
3. Left - Extracts a specified number of characters from the beginning of a text.
4. Right - Extracts a specified number of characters from the end of a text.
5. Concatenation - Combines two or more text values into one.
6. Trim - Removes extra spaces from the beginning and end of a text.
7. Character Length - Counts the number of characters in a text.
8. MID - Extracts a portion of text from the middle of a longer text.
9. Length - Counts the total number of characters in a text.

- 1. Lowercase (LOWER)

  Convert the Stud_Name column to lowercase:

  Select LOWER(Stud_Name) as Lowercase_Name from student_information;

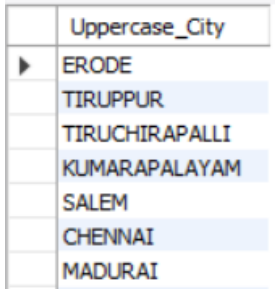


Lowercase_Name
vasanth
guru
gokul
mani
moorthy
amutha
jaga

- 2. Uppercase (UPPER)

  Convert the City_State column to uppercase:

  Select UPPER(City_State) as Uppercase_City from student_information;

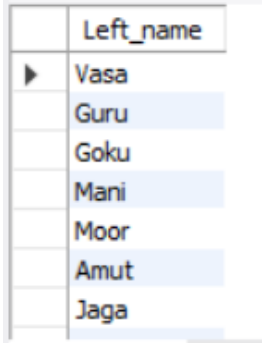


Uppercase_City
ERODE
TIRUPPUR
TIRUCHIRAPALLI
KUMARAPALAYAM
SALEM
CHENNAI
MADURAI

- 3. LEFT

  Retrieve the leftmost 4 characters of the Stud_Name column:

  Select LEFT(Stud_Name, 4) as Left_name from student_information;

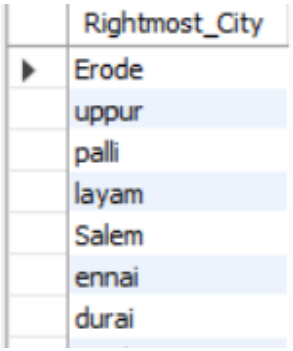


Left_name
Vasa
Guru
Goku
Mani
Moor
Amut
Jaga

- 4. RIGHT

  Retrieve the rightmost 5 characters of the City_State column:

  Select RIGHT(City_State, 5) as Rightmost_City from student_information;

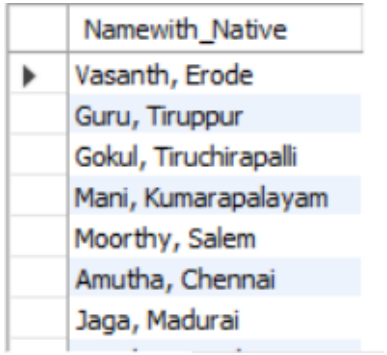


Rightmost_City
Erode
uppur
palli
layam
Salem
ennai
durai

- 5. Concatenation (CONCAT)

  Combine `Stud_Name` and `City_State` into a single column:

  Select CONCAT(Stud_Name, ', ', City_State) as Namewith_Native from student_information;

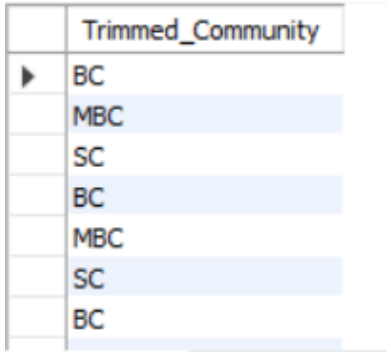


Namewith_Native
Vasanth, Erode
Guru, Tiruppur
Gokul, Tiruchirapalli
Mani, Kumarapalayam
Moorthy, Salem
Amutha, Chennai
Jaga, Madurai

- 6. TRIM

  Remove leading and trailing spaces from the `Community` column:

  Select TRIM(Community) as Trimmed_Community from student_information;

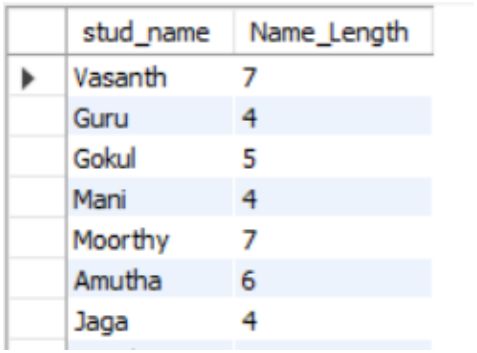


Trimmed_Community
BC
MBC
SC
BC
MBC
SC
BC

- 7. Character Length (CHAR_LENGTH)

  Calculate the character length of the Stud_Name column:

  Select stud_name, CHAR_LENGTH(Stud_Name) as Name_Length from student_information;

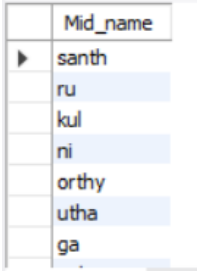


stud_name	Name_Length
Vasanth	7
Guru	4
Gokul	5
Mani	4
Moorthy	7
Amutha	6
Jaga	4

- **8. MID**

  Retrieve a Mid_name of `stud_Name` starting from the 3rd character and 5 characters long:

  Select MID(stud_name, 3, 5) as Mid_name from student_information;



Mid_name
santh
ru
kul
ni
orthy
utha
ga

- 9. Length

  Calculate the length (in characters) of the Community column:

  Select community, LENGTH(Community) as Community_Length from student_information;

	community	Community_Length
▶	BC	2
	MBC	3
	SC	2
	BC	2
	MBC	3
	SC	2
	BC	2

LOGICAL FUNCTIONS

- **Logical Functions in MySQL**

  Logical Functions help with conditional operations.

1. IF - Makes decisions based on conditions.
2. IF with AND - It's like combining conditions with "and"; both conditions must be true for an action.
3. IF with OR - It's like combining conditions with "or"; one of the conditions must be true for an action.
4. COUNT IF - It's like counting data based on specific conditions; count only if a condition is met.

- **1. IF Function**

  Use the IF function to classify students as ‘Pass’ or ‘Fail’ based on their Finance marks (e.g., considering a passing score of 35):

  Select Stud_ID, Finance, IF(Finance >= 35, 'Pass', 'Fail') as Pass_Status from Marks_info;

Stud_ID	Finance	Pass_Status
1	75	Pass
2	92	Pass
3	38	Pass
4	39	Pass
5	34	Fail
6	44	Pass
7	50	Pass

- **2. IF with AND Conditions**

  Classify students as ‘Pass’ if they scored 35 or above in Finance, Cost and corporate:

  Select Stud_ID, Finance, Cost, corporate, IF(Finance >= 35 AND Cost >= 35 and corporate>=35, 'Pass’ , 'Fail') as Status from Marks_info;

Stud_ID	Finance	Cost	corporate	Status
1	75	76	65	Pass
2	92	90	19	Fail
3	38	37	46	Pass
4	39	90	58	Pass
5	34	89	20	Fail
6	44	38	60	Pass
7	50	26	98	Fail

- 3. IF with OR Conditions

  Classify students as ‘Pass’ if they scored 60 or above in either ‘Finance’ or ‘Cost’:

  Select Stud_ID, Finance, Cost, IF(Finance >= 60 OR Cost >= 60, 'Pass', 'Fail') as Pass_Status from Marks_info;

Stud_ID	Finance	Cost	Pass_Status
1	75	76	Pass
2	92	90	Pass
3	38	37	Fail
4	39	90	Pass
5	34	89	Pass
6	44	38	Fail
7	50	26	Fail

- **4. COUNT**

  Count the number of students who passed Finance with a score of 70 or above:

  Select COUNT(*) as Passed_Finance_Count from Marks_info where Finance >= 70;

Passed_Finance_Count
10

RDBMS SYSTEMS

RDBMS SYSTEMS

- RDBMS systems efficiently organize data.
- Two Table Connections - Link data between two tables using keys, like connecting students with their courses.
- Three Table Connections - Extend connections to three tables, enabling more complex relationships, such as students, courses, and marks.

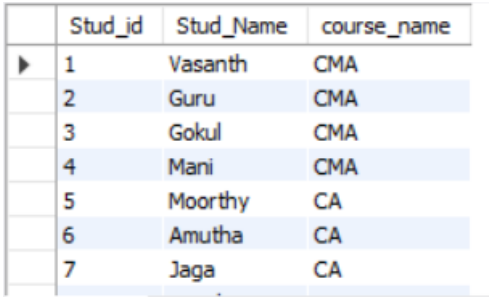
- 1. Two Table Connections

Connect the student_information and course_information tables to retrieve information about students and the courses they are enrolled in. The common field connecting these tables is Course_ID.

```
Select student_information.Stud_id, student_information.Stud_Name, course_information.course_name

from student_information

INNER JOIN course_information ON student_information.Course_ID = course_info.course_id;
```



Stud_id	Stud_Name	course_name
1	Vasanth	CMA
2	Guru	CMA
3	Gokul	CMA
4	Mani	CMA
5	Moorthy	CA
6	Amutha	CA
7	Jaga	CA

In this query, we are using an INNER JOIN to combine data from the student_information and Course_information tables based on the matching Course_ID and course_id. This retrieves student names and the corresponding course names they are enrolled in.

- 2. Three Table Connections

  let's expand on the previous example and add the marks_info table to retrieve students, their enrolled courses, and their marks.

  Select student_information.Stud_id, student_information.Stud_Name, Course_information.course_name, marks_info.Finance, marks_info.Cost, marks_info.corporate

  from student_information

  INNER JOIN Course_information ON student_information.Course_ID = Course_information.course_id

  INNER JOIN marks_info ON student_information.Stud_id = marks_info.Stud_ID;

Stud_id	Stud_Name	course_name	Finance	Cost	corporate
1	Vasanth	CMA	75	76	65
2	Guru	CMA	92	90	19
3	Gokul	CMA	38	37	46
4	Mani	CMA	39	90	58
5	Moorthy	CA	34	89	20
6	Amutha	CA	44	38	60
7	Jaga	CA	50	26	98

In this query, we are using two INNER JOIN operations. The first JOIN connects student_information and Course_information based on the Course_ID and course_id, and the second JOIN connects student_information and marks_info based on the Stud_id.

JOIN QUERIES

Join Queries in MySQL

- Join queries combine data from multiple tables.

1. INNER JOIN - Combines matching data.
2. LEFT JOIN - Retrieves all from the left and matching from the right.
3. RIGHT JOIN - Retrieves all from the right and matching from the left.
4. CROSS JOIN - Combines all rows from both tables.
5. FULL OUTER JOIN - Retrieves all data from both tables.
6. CASE WHEN THEN END - Makes conditional choices.
7. Double CASE with END - Nested conditional choices.
8. CASE with AND - Combines CASE with AND conditions.
9. CASE with OR - Combines CASE with OR conditions.
10. HAVING - Filters grouped data.
11. JOIN Triggers - Join tables using triggers.

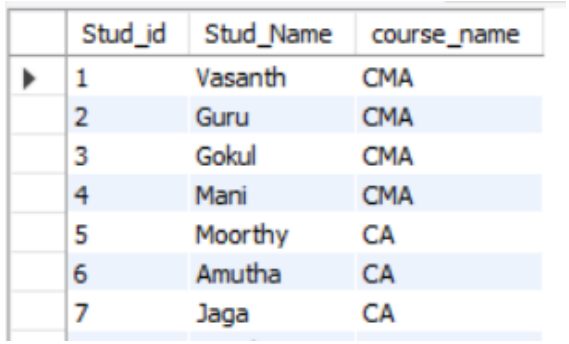
- **1. INNER JOIN**

  If you want to retrieve information about students and the courses they are enrolled in. Use an INNER JOIN to get only the matching records.

  Select student_information.Stud_id, student_information.Stud_Name, Course_information.course_name

  from student_information

  INNER JOIN Course_information ON student_information.Course_ID = Course_information.course_id;



Stud_id	Stud_Name	course_name
1	Vasanth	CMA
2	Guru	CMA
3	Gokul	CMA
4	Mani	CMA
5	Moorthy	CA
6	Amutha	CA
7	Jaga	CA

This query retrieves student names and the corresponding course names they are enrolled in, but only for students who are enrolled in courses.

- **2. LEFT JOIN**
- If you want to retrieve all students, including those not enrolled in any courses, you can use a LEFT JOIN:
- I inserted a dummy row in the student_information to show the difference of the left join

```
insert into student_information (stud_id, stud_name, city_state, age, roll_no, community)values
(29,'Praveen','Mettur',26,13029,'MBC');

Select student_information.Stud_id, student_information.Stud_Name,    Course_information.course_name

from student_information

LEFT JOIN Course_information ON student_information.Course_ID = Course_information.course_id;
```

Stud_id	Stud_Name	course_name
23	Veera	CA
24	Devi	CS
25	Devan	CMA
26	Keerthi	CA
27	Venkatesh	CMA
28	Raja	CS
29	Praveen	NULL

This query retrieves all student names and their corresponding course names matching right table. Students without courses will still appear in the result with NULL values in the course name column.

- **3. RIGHT JOIN**

Similarly, you can use a RIGHT JOIN to retrieve all course names, including those without enrolled students:

I inserted a dummy row in the Course_information to show the difference of the Right join

insert into Course_information values (4,'SQL');

Select student_information.Stud_id, student_information.Stud_Name, Course_information.course_name

from student_information

RIGHT JOIN Course_information ON student_information.Course_ID = Course_information.course_id;

Stud_id	Stud_Name	course_name
18	Selvi	CS
13	Arun	CS
12	Suja	CS
11	Manasi	CS
10	Kabilan	CS
9	Arthi	CS
NULL	NULL	SQL

This query retrieves all course names and the corresponding student names if available. Courses without students will still appear in the result with NULL values in the student name column.

- **4. CROSS JOIN**
- A CROSS JOIN combines all rows from one table with all rows from another table. It doesn't require a specific common field:

```
Select student_information.Stud_id, student_information.Stud_Name, Course_information.course_name

from student_information

CROSS JOIN Course_information;
```

Stud_id	Stud_Name	course_name
1	Vasanth	SQL
1	Vasanth	CS
1	Vasanth	CA
1	Vasanth	CMA
2	Guru	SQL
2	Guru	CS
2	Guru	CA

This query retrieves all possible combinations of student names and course names.

5. FULL OUTER JOIN

- MySQL doesn't support FULL OUTER JOIN directly, but you can emulate it with a combination of LEFT JOIN and RIGHT JOIN through UNION:

```
Select student_information.Stud_id, student_information.Stud_Name, Course_information.course_name

from student_information

LEFT JOIN Course_information ON student_information.Course_ID = Course_information.course_id

UNION

select student_information.Stud_id, student_information.Stud_Name, Course_information.course_name

from student_information

RIGHT JOIN Course_information ON student_information.Course_ID = Course_information.course_id;
```

Stud_id	Stud_Name	course_name
24	Devi	CS
25	Devan	CMA
26	Keerthi	CA
27	Venkatesh	CMA
28	Raja	CS
29	Praveen	NULL
NULL	NULL	SQL

This query retrieves all student names and course names, including those without matches in both tables.

6. CASE WHEN THEN END

- This query uses the ‘CASE WHEN THEN END’ construct to categorize students based on their age.

```
Select Stud_id, Stud_Name, age,
    CASE
        WHEN Age >= 20 THEN 'Old'
        WHEN Age < 20 THEN 'Young'
        ELSE 'Unknown'
    END as Age_Group
from student_information;
```

Stud_id	Stud_Name	age	Age_Group
1	Vasanth	21	Old
2	Guru	20	Old
3	Gokul	18	Young
4	Mani	24	Old
5	Moorthy	18	Young
6	Amutha	17	Young
7	Jaga	24	Old
8	Pavithra	23	Old
9	Arthi	17	Young
10	Kabilan	24	Old
11	Manasi	17	Young
12	Suja	23	Old
13	Arun	22	Old
14	Deepa	20	Old
15	Sindhu	22	Old
16	Madhavi	20	Old
17	Swetha	17	Young
18	Selvi	22	Old
19	Pooja	19	Young
20	Lakshmi	17	Young
21	Veeramani	21	Old
22	Pandian	21	Old
23	Veera	20	Old
24	Devi	20	Old
25	Devan	21	Old

- 7. Double CASE with END Statement
- This query uses nested ‘CASE’ statements to categorize students based on age and roll number.

```
Select Stud_id, Stud_Name, age, roll_no,
    CASE
        WHEN Age >= 20 THEN
            CASE
                WHEN Roll_no < 13010 THEN 'Senior'
                ELSE 'Junior'
            END
        ELSE 'Unknown'
    END AS Student_Status
from student_information;
```

Stud_id	Stud_Name	age	roll_no	Student_Status
1	Vasanth	21	13001	Senior
2	Guru	20	13002	Senior
3	Gokul	18	13003	Unknown
4	Mani	24	13004	Senior
5	Moorthy	18	13005	Unknown
6	Amutha	17	13006	Unknown
7	Jaga	24	13007	Senior
8	Pavithra	23	13008	Senior
9	Arthi	17	13009	Unknown
10	Kabilan	24	13010	Junior
11	Manasi	17	13011	Unknown
12	Suja	23	13012	Junior
13	Arun	22	13013	Junior
14	Deepa	20	13014	Junior
15	Sindhu	22	13015	Junior
16	Madhavi	20	13016	Junior
17	Swetha	17	13017	Unknown
18	Selvi	22	13018	Junior
19	Pooja	19	13019	Unknown
20	Lakshmi	17	13020	Unknown
21	Veeramani	21	13021	Junior
22	Pandian	21	13022	Junior
23	Veera	20	13023	Junior

- **8. CASE with AND Statement**
- This query uses ‘CASE’ with an ‘AND’ condition to categorize students based on age and roll number.

```
Select Stud_id, Stud_Name, age, roll_no,
    CASE
        WHEN Age >= 20 AND Roll_no > 13010 THEN 'Senior'
        WHEN Age < 20 AND Roll_no <= 13010 THEN 'Junior'
        ELSE 'Unknown'
    END AS Student_Status
from student_information;
```

Stud_id	Stud_Name	age	roll_no	Student_Status
1	Vasanth	21	13001	Unknown
2	Guru	20	13002	Unknown
3	Gokul	18	13003	Junior
4	Mani	24	13004	Unknown
5	Moorthy	18	13005	Junior
6	Amutha	17	13006	Junior
7	Jaga	24	13007	Unknown
8	Pavithra	23	13008	Unknown
9	Arthi	17	13009	Junior
10	Kabilan	24	13010	Unknown
11	Manasi	17	13011	Unknown
12	Suja	23	13012	Senior
13	Arun	22	13013	Senior
14	Deepa	20	13014	Senior
15	Sindhu	22	13015	Senior
16	Madhavi	20	13016	Senior
17	Swetha	17	13017	Unknown
18	Selvi	22	13018	Senior
19	Pooja	19	13019	Unknown
20	Lakshmi	17	13020	Unknown
21	Veeramani	21	13021	Senior
22	Pandian	21	13022	Senior
23	Veera	20	13023	Senior
24	Devi	20	13024	Senior
25	Devan	21	13025	Senior

- **9. CASE with OR Statement**
- This query uses ‘CASE’ with an ‘OR’ condition to categorize students based on age and roll number.

```
Select Stud_id, Stud_Name, age, roll_no,
    CASE
        WHEN Age >= 20 OR Roll_no > 13010 THEN 'Senior'
        WHEN Age < 20 OR Roll_no <= 13010 THEN 'Junior'
        ELSE 'Unknown'
    END AS Student_Status
from student_information;
```

Stud_id	Stud_Name	age	roll_no	Student_Status
1	Vasanth	21	13001	Senior
2	Guru	20	13002	Senior
3	Gokul	18	13003	Junior
4	Mani	24	13004	Senior
5	Moorthy	18	13005	Junior
6	Amutha	17	13006	Junior
7	Jaga	24	13007	Senior
8	Pavithra	23	13008	Senior
9	Arthi	17	13009	Junior
10	Kabilan	24	13010	Senior
11	Manasi	17	13011	Senior
12	Suja	23	13012	Senior
13	Arun	22	13013	Senior
14	Deepa	20	13014	Senior
15	Sindhu	22	13015	Senior
16	Madhavi	20	13016	Senior
17	Swetha	17	13017	Senior
18	Selvi	22	13018	Senior
19	Pooja	19	13019	Senior
20	Lakshmi	17	13020	Senior
21	Veeramani	21	13021	Senior
22	Pandian	21	13022	Senior
23	Veera	20	13023	Senior
24	Devi	20	13024	Senior
25	Devan	21	13025	Senior

- **10. Having**
- This query uses ‘CASE’ with ‘HAVING’ to categorize students based on city_state from salem and Average Marks above 60%

```
Select s.stud_id, s.stud_name, s.city_state, s.age, s.community, c.course_id, c.course_name,m.finance, m.cost,
m.corporate, (m.finance+ m.cost+ m.corporate) as Total_marks,

concat(round((m.finance+ m.cost+ m.corporate)/3,0),"%") as Average_marks

from student_information as s INNER JOIN Course_information as c on s.course_id = c.course_id

INNER JOIN marks_info as m on s.stud_id = m.stud_id

Where city_state = 'salem'

Having average_marks >60 ;
```

stud_id	stud_name	city_state	age	community	course_id	course_name	finance	cost	corporate	Total_marks	Average_marks
19	Pooja	Salem	19	BC	1	CMA	55	76	76	207	69%
26	Keerthi	Salem	17	MBC	2	CA	48	66	70	184	61%

STORED PROCEDURES

Stored Procedures in MySQL

- Stored procedures are like predefined scripts that can be executed with a single call, making complex tasks easier to manage in MySQL.

1. CREATE PROCEDURE - Defines a reusable set of SQL statements.
2. BEGIN-END - Encloses the code block of the stored procedure.
3. CALL PROCEDURE – Executing the stored procedure.
4. DROP PROCEDURE - Removes a stored procedure from the database.

- **CREATING THE STORED PROCEDURE:**

```
DELIMITER //
CREATE PROCEDURE average_marks()
BEGIN
Select s.stud_id, s.stud_name, s.city_state, s.age, s.community, c.course_id, c.course_name,m.finance, m.cost,
m.corporate,     (m.finance + m.cost + m.corporate) as Total_marks,
Round((m.finance + m.cost + m.corporate) / 3, 0) as Average_marks
from student_information as s INNER JOIN Course_information as c ON s.course_id = c.course_id
INNER JOIN marks_info as m ON s.stud_id = m.stud_id;
END //
DELIMITER ;
```

- The stored procedure for average marks for students has been successfully created.

- **Call the Stored Procedure:**

```
CALL average_marks();
```

This executes the average_marks stored procedure and displays the results.

stud_id	stud_name	city_state	age	community	course_id	course_name	finance	cost	corporate	Total_marks	Average_marks
1	Vasanth	Erode	21	BC	1	CMA	75	76	65	216	72
2	Guru	Tiruppur	20	MBC	1	CMA	92	90	19	201	67
3	Gokul	Tiruchirapalli	18	SC	1	CMA	38	37	46	121	40
4	Mani	Kumarapalayam	24	BC	1	CMA	39	90	58	187	62
5	Moorthy	Salem	18	MBC	2	CA	34	89	20	143	48
6	Amutha	Chennai	17	SC	2	CA	44	38	60	142	47
7	Jaga	Madurai	24	BC	2	CA	50	26	98	174	58
8	Pavithra	Erode	23	MBC	2	CA	59	78	82	219	73
9	Arthi	Tiruppur	17	SC	3	CS	89	47	88	224	75
10	Kabilan	Tiruchirapalli	24	BC	3	CS	20	25	100	145	48

Result 4

TRIGGERS

Triggers in MYSQL

It is a special type of stored procedure that is invoked automatically in response to an event. Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE

We can define the maximum six types of actions or events in the form of triggers

1. Before Insert - It is activated before the insertion of data into the table.
2. After Insert - It is activated after the insertion of data into the table.
3. Before Update - It is activated before the update of data in the table.
4. After Update - It is activated after the update of the data in the table.
5. Before Delete - It is activated before the data is removed from the table.
6. After Delete - It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

Before Insert

- Created a new column to view the trigger result
- alter table marks_info add column result varchar(20);

```
delimiter //
create trigger result before insert on marks_info for each row
Begin
if new.finance and new.cost and new.corporate  >=35 then set new.result = 'Pass’;
else set new.result = 'RA’;
END IF;
end //
delimiter ;
```

Marks_ID	Stud_ID	finance	Cost	Corporate	result
14001	1	75	76	65	Pass
14002	2	92	90	19	RA
14003	3	38	37	46	Pass
14004	4	39	90	58	Pass
14005	5	34	89	20	RA
14006	6	44	38	60	Pass
14007	7	50	26	98	Pass
14008	8	59	78	82	Pass
14009	9	89	47	88	Pass
14010	10	20	25	100	Pass
14011	11	74	50	100	Pass
14012	12	81	62	31	RA
NULL	NULL	NULL	NULL	NULL	NULL

THANK YOU