MYSQL PROJECT

By,
Bhavani

MYSQL

- MYSQL IS THE MOST POPULAR OPEN SOURCE RELATIONAL DATABASE MANAGEMENT SYSTEM.
- MYSQL USED FOR DEVELOPING VARIOUS WEB BASED SOFTWARE DEVELOPMENT
- DEVELOPED BY COMPANY MYSQL AB. (BASED ON CNC++)

MYSQL WORKBENCH

- IT IS A VISUAL DATABASE DESIGN TOOL THAT INTEGRATES SQL DEVELOPMENT, ADMINISTRATION, DATABASE DESIGN, CREATION AND MAINTENANCE INTO A SINGLE INTEGRATED DEVELOPMENT ENVIRONMENT FOR THE MYSQL DATABASE SYSTEM

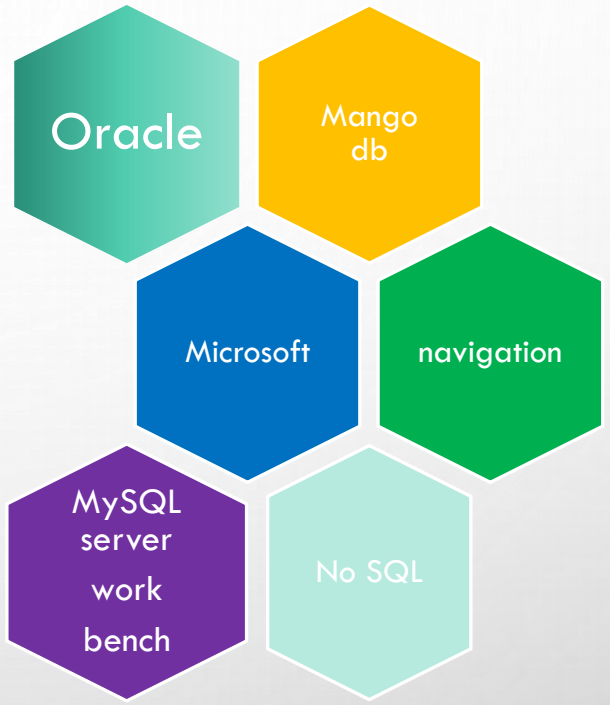
- **MYSQL WORKBENCH 8.0**. CE VERSION

FEATURES OF MYSQL:

- RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)
- EASY TO USE
- STORING DATA IS SECURE
- OPEN SOURCE & FREE TO DOWNLOAD
- COMPATIBLE ON MANY OPERATING SYSTEMS.

MYSQL SERVERS:

- MYSQL Server Workbench
- oracle server
- mango dB server
- no SQL
- navigation server
- Microsoft server



COMMANDS:

❖Data Query Language(DQL) Commands

❖Data Definition Language (DDL) commands

❖Data Manipulation Language (DML) commands

DATABASE

DEFINITION:

- A DATABASE IS AN ORGANIZED COLLECTION OF STRUCTURED INFORMATION, OR DATA, TYPICALLY STORED ELECTRONICALLY IN A COMPUTER SYSTEM.

- A DATABASE IS USUALLY CONTROLLED BY A DATABASE MANAGEMENT SYSTEM (DBMS).

- THE DATA CAN THEN BE EASILY ACCESSED, MANAGED, MODIFIED, UPDATED, CONTROLLED, AND ORGANIZED.

- MOST DATABASES USE STRUCTURED QUERY LANGUAGE (SQL) FOR WRITING AND QUERYING DATA.

DBMS (VS) RDBMS

DBMS	RDBMS
• DBMS stands for "Database Management System".	• RDBMS stands for "Relational Database Management System".
• DBMS technology stores the data in the form of files.	• RDBMS stores the data in the form of tables.
• DBMS is designed to handle small amounts of data.	• RDBMS is designed to deal with vast amount of data.
• DBMS provides support only for a single user at a time.	• RDBMS provides support for multiple users at a time.

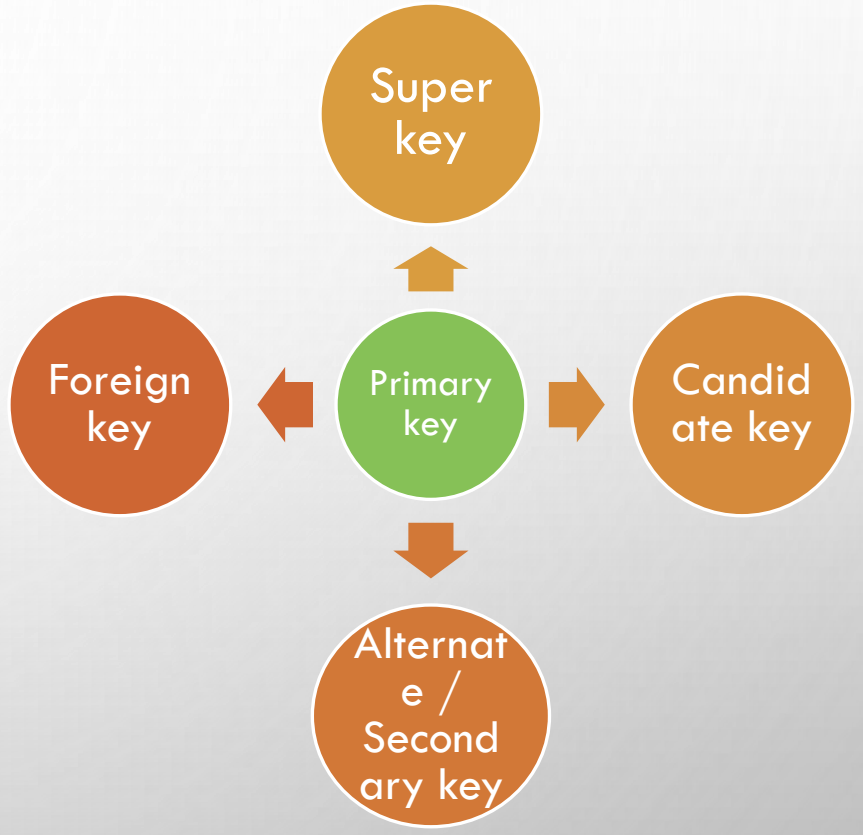
DBMS KEYS

PRIMARY KEY:

- for identify the row
- unique key
- duplicates cannot be allowed
- cannot be null (No null values)
- Eg: (Emp_ID)

SUPER KEY:

- a single key attribute
- a group of multiple keys
- These keys have redundant attributes along with a key attribute. *Eg:* (Emp_ID, Name)



CANDIDATE KEY

- MINIMAL SUPER KEY IS TERMED AS THE CANDIDATE KEY.
- CANDIDATE KEYS SHOULD ALWAYS BE UNIQUE, CAN HAVE NULL VALUES.
- HAVE A SINGLE ATTRIBUTE
- A TABLE MAY HAVE ONE OR MORE CANDIDATE KEYS

  EG:(`EMP_ID`),

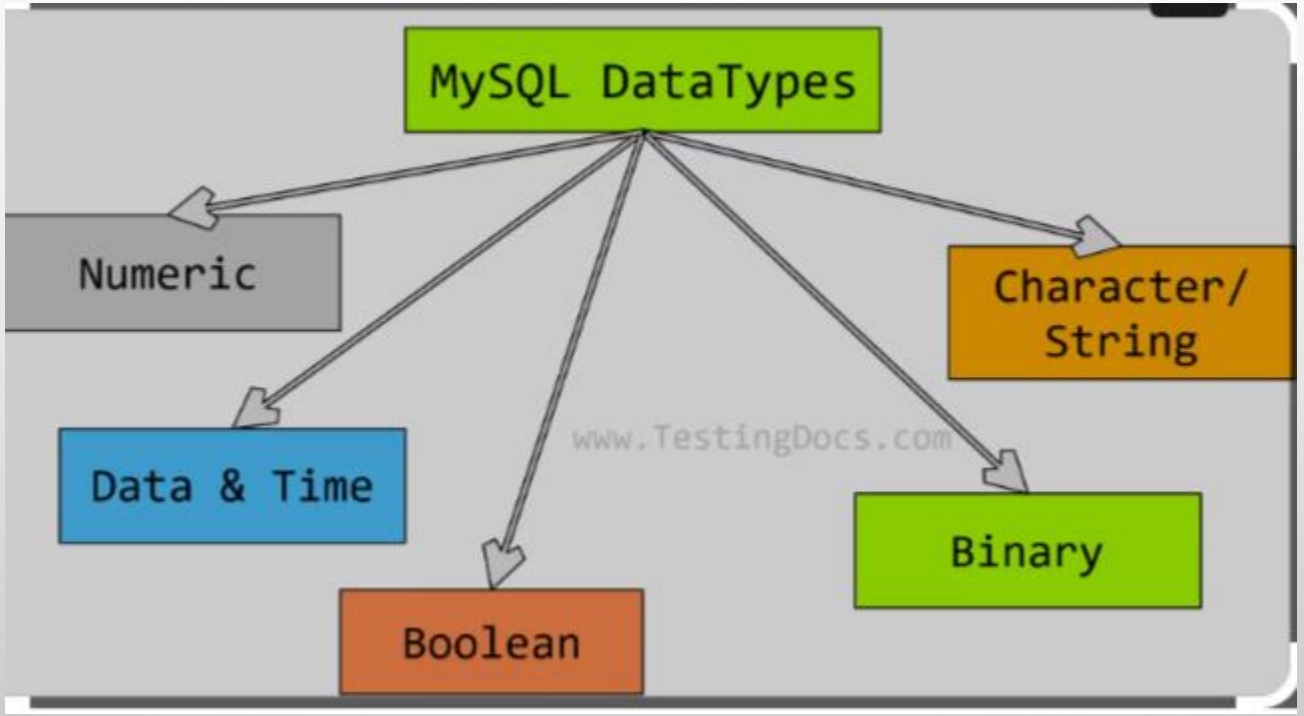
Foreign Key

- A foreign key is a reference key. foreign key is a key used to link two tables together.
- Foreign key is used to maintain relationship between two tables.
- (*Example*: Reg_No of Hostel Table is the Primary Key of the Student Table)

ALTERNATE KEY

- THE KEYS OTHER THAN PRIMARY KEY ARE CALLED ALTERNATE KEYS
- ALTERNATE(*SECONDARY*) KEYS CAN BE MADE PRIMARY KEY.
- EXAMPLE: (`EMAIL_ID`), SOMETIMES SECONDARY/ALTERNATE KEY IS REQUIRED FOR THE INDEXING, FOR BETTER AND FASTER SEARCHING.

DATA TYPES


MySQL DataTypes
Numeric
Character/
String
Data & Time
www.TestingDocs.com
Binary
Boolean

NUMERIC DATA TYPE

- ***INT*** :INTEGER DATA TYPE FOR STORING WHOLE NUMBERS.
- ***BIGINT*** :LARGE INTEGER DATA TYPE FOR VERY LARGE WHOLE NUMBERS.
- ***SMALLINT*** :SMALL INTEGER DATA TYPE FOR SMALL WHOLE NUMBERS.
- ***DECIMAL/NUMERIC*** :FIXED-POINT NUMERIC DATA TYPE THAT STORES EXACT NUMERIC VALUES WITH A SPECIFIED PRECISION AND SCALE.
- ***FLOAT/REAL*** : FLOATING-POINT NUMERIC DATA TYPE FOR APPROXIMATE NUMERIC VALUES WITH A SPECIFIED PRECISION.

DATA QUERY LANGUAGE (DQL) COMMANDS

- ***SELECT-*** RETRIEVES DATA FROM ONE OR MORE TABLES.
- ***FROM***- SPECIFIES THE TABLE(S) FROM WHICH TO RETRIEVE DATA.
- ***WHERE-***FILTERS DATA BASED ON SPECIFIED CONDITIONS.
- ***GROUP BY***-GROUPS DATA BASED ON ONE OR MORE COLUMNS.
- ***HAVING-***FILTERS GROUPED DATA BASED ON CONDITIONS.
- ***ORDER BY***-SORTS THE RESULT SET BASED ON ONE OR MORE COLUMNS.
- ***JOIN-***COMBINES ROWS FROM TWO OR MORE TABLES BASED ON A RELATED COLUMN.

CONTENTS

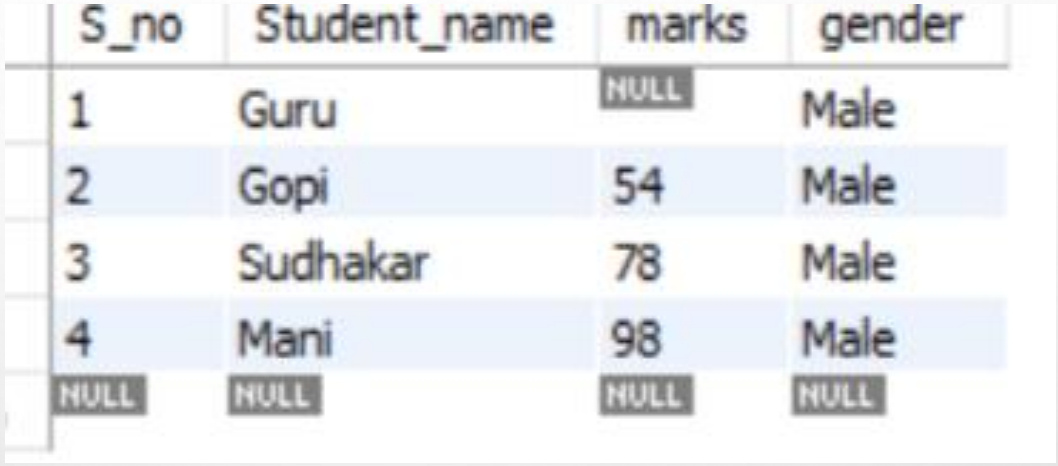
My SQL Main Commands

- - Create database - (Create a New Database)
- - Show Databases - (View Databases)
- - Drop Database- Deletes a database
- -Select Databases- extract data from a database
- - Alter Database - (Modify Database)
- - Create Tables – Creates a new table
- - Show tables - View tables
- - Insert Values – Insert new data into a database

- - DROP TABLE – DELETES A TABLE
- - ALTER TABLE (FOR NEW COLUMN CREATION)
- - ALTER TABLE –MODIFIES THE TABLE
- - ALTER TABLE DROP (DROP THE COLUMN)
- - ALTER TABLE RENAME (RENAME THE TABLE)
- - UPDATE TABLE (TO CHANGE THE VALUES)
- - DELETE STATEMENT – REMOVE RECORDS FROM THE *MYSQL* TABLE

CONSTRAINTS

- NOT NULL
- AUTO INCREMENT
- PRIMARY KEY
- FOREIGN KEY
- DEFAULT



S_no	Student_name	marks	gender
1	Guru	NULL	Male
2	Gopi	54	Male
3	Sudhakar	78	Male
4	Mani	98	Male
NULL	NULL	NULL	NULL

QUERY: CREATE TABLE STUDENT_DET (S_NO INT AUTO_INCREMENT, STUDENT_NAMEVARCHAR(20) NOT NULL, MARKS INT, GENDER VARCHAR(20),PRIMARY KEY(S_NO));

TABLE CREATION

```
CREATE TABLE  STUDENT _INFO (STUDENT_ID INT, STUDENT_NAME VARCHAR(20), CITY_STATE VARCHAR(20),
AGE INT, RESULTS VARCHAR (20), MARKS INT,PRIMARY KEY(STUDENT_ID));
INSERT INTO STUDENT_INFO VALUES
(1,     'GEETHA',       'ERODE',  21,     'NO RANK',      37),
(2,     'GURU',   'TIRUPPUR',     20,     'NO RANK',      28),
(3,     'GOKUL', 'TIRUCHIRAPALLI', 18,    'AVERAGE',      40),
(4,     'MANI',   'KUMARAPALAYAM',      24,     'NO RANK',      31),
(5,     'MOORTHY',      'SALEM', 18,     'VERY GOOD',    86);
SELECT * FROM EMP_INFO;
```

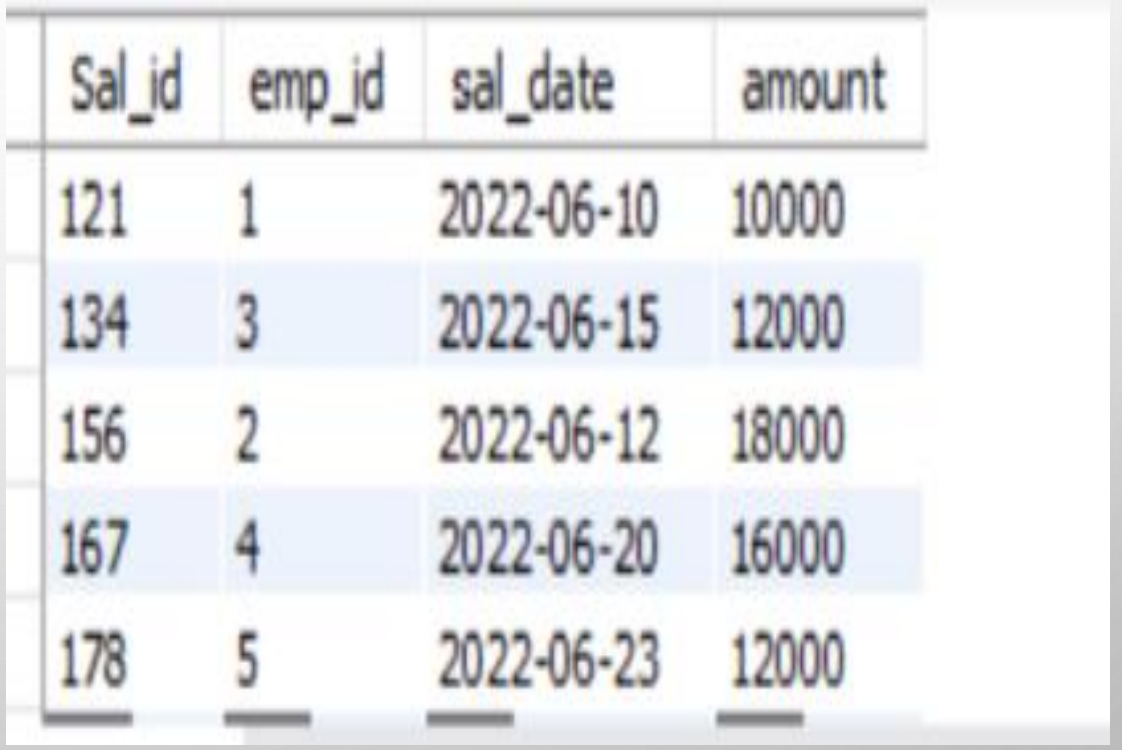
OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
1	Geetha	Erode	21	No Rank	37
2	Guru	Tiruppur	20	No Rank	28
3	Gokul	Tiruchirapalli	18	Average	40
4	Mani	Kumarapalayam	24	No Rank	31
5	Moorthy	Salem	18	Very Good	86
6	Amutha	Chennai	17	Average	61
7	Jaga	Madurai	24	Very Good	89
8	Pavithra	Erode	23	Average	68
9	Arthi	Tiruppur	17	Average	53
10	Kabilan	Tiruchirapalli	24	Average	67
11	Manasi	Kumarapalayam	17	Excellent	97
12	Suja	Salem	23	Very Good	85

TABLE 2

```
CREATE TABLE SAL_DET (SAL_ID INT, EMP_ID INT,
SAL_DATE DATE, AMOUNT INT, PRIMARY KEY(SAL_ID));
INSERT INTO SAL_DET VALUES
(121,    1,        '2022-06-10',    10000),
(156,    2,        '2022-06-12',    18000),
(134,    3,        '2022-06-15',    12000),
(167,    4,        '2022-06-20',    16000),
(178,    5,        '2022-06-23',    12000);
 SELECT * FROM SAL_DET;
```

OUTPUT:



Sal_id	emp_id	sal_date	amount
121	1	2022-06-10	10000
134	3	2022-06-15	12000
156	2	2022-06-12	18000
167	4	2022-06-20	16000
178	5	2022-06-23	12000

MY SQL GENERAL FUNCTIONS

- Where
- Or
- And
- In
- Not in
- >=
- <=
- <>(Not in)
- !
- Count
- Distinct

- Count with Distinct
- Order by ASC
- Order by DESC
- Group by
- Limit
- Desc limit
- Like
- Not like
- Between

Where:

WHERE ***clause*** is used to filter records.

It is used to extract only those records that fulfill a specified ***condition***

QUERY: Select * from Student_info where City_state = 'Chennai';

OUTPUT:

	Student_id	Student_name	City_state	age	Results	marks
▶	6	Amutha	Chennai	17	Average	61
	13	Arun	Chennai	22	No Rank	32
	20	Lakshmi	Chennai	17	Excellent	99
	27	Venkatesh	Chennai	24	Good	75
*	NULL	NULL	NULL	NULL	NULL	NULL

AND: The AND operator displays a record if all the conditions are TRUE.

QUERY: select * from student_info where City_state = 'madurai' and Age= 20;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
14	Deepa	Madurai	20	Average	49
NULL	NULL	NULL	NULL	NULL	NULL

OR:

The OR ***operator displays a record if any of the conditions are TRUE.***

QUERY: select * from Student_info where city_state = 'chennai' or Age= 20;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
2	Guru	Tiruppur	20	No Rank	28
6	Amutha	Chennai	17	Average	61
13	Arun	Chennai	22	No Rank	32
14	Deepa	Madurai	20	Average	49
16	Madhavi	Tiruppur	20	Good	78

LESSER THAN

Lesser than (<) ***operator*** The ***less than operator*** is used to test whether an expression (or number) is ***less than*** another one.

QUERY: select * from Student_info where age < 23;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
1	Geetha	Erode	21	No Rank	37
2	Guru	Tiruppur	20	No Rank	28
3	Gokul	Tiruchirapalli	18	Average	40
5	Moorthy	Salem	18	Very Good	86
6	Amutha	Chennai	17	Average	61

LESS THAN OR EQUAL TO : The lesser than equal to(<=) shows the lower values and also equal to that values

QUERY: select * from Student_info where marks<=90;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
1	Geetha	Erode	21	No Rank	37
2	Guru	Tiruppur	20	No Rank	28
3	Gokul	Tiruchirapalli	18	Average	40
4	Mani	Kumarapalayam	24	No Rank	31
5	Moorthy	Salem	18	Very Good	86
6	Amutha	Chennai	17	Average	61
7	Jaga	Madurai	24	Very Good	89
8	Pavithra	Erode	23	Average	68

IN :

IN operator with the WHERE clause to match values in a list.
QUERY: select * from Student_info where city_state in ('chennai', 'Erode');
OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
1	Geetha	Erode	21	No Rank	37
6	Amutha	Chennai	17	Average	61
8	Pavithra	Erode	23	Average	68
13	Arun	Chennai	22	No Rank	32
15	Sindhu	Erode	22	Average	65

NOT IN:

NOT IN operator excludes the rows that match values in the list. It returns all the rows except the excluded rows.
QUERY: select * from Student_info where city_state not in ('chennai', 'madurai');
OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
1	Geetha	Erode	21	No Rank	37
2	Guru	Tiruppur	20	No Rank	28
3	Gokul	Tiruchirapalli	18	Average	40
4	Mani	Kumarapalayam	24	No Rank	31
5	Moorthy	Salem	18	Very Good	86

Count: The COUNT() function returns the number of records returned by a select query.
QUERY: select count(Student_name) from Student_info;
Output:

count(Student_name)
28

- Distinct: A distinct operator removes duplicate rows from a data set.
QUERY: select distinct city_state from Student_info;
Output:

city_state
Erode
Tiruppur
Tiruchirapalli
Kumarapalayam
Salem
Chennai
Madurai

Count With Distinct:
The **COUNT DISTINCT** function returns the number of unique values in the column or expression,
QUERY: select count(Distinct City_state) From Student_info;
Output:

count(Distinct City_state)
7

ORDER BY ASC:

THE ORDER BY KEYWORD SORTS THE RECORDS IN ASCENDING ORDER BY DEFAULT. WHICH IS SMALLEST TO LARGEST.

QUERY: SELECT * FROM STUDENT_INFO ORDER BY STUDENT_NAME ASC;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
6	Amutha	Chennai	17	Average	61
9	Arthi	Tiruppur	17	Average	53
13	Arun	Chennai	22	No Rank	32
14	Deepa	Madurai	20	Average	49
25	Devan	Kumarapalayam	21	Excellent	100
24	Devi	Tiruchirapalli	20	Excellent	96
1	Geetha	Erode	21	No Rank	37
3	Gokul	Tiruchirapalli	18	Average	40
2	Guru	Tiruppur	20	No Rank	28

ORDER BY DESC:

TO SORT THE RECORDS IN DESCENDING ORDER, USE THE DESC KEYWORD. WHICH IS LARGEST TO SMALLEST

QUERY: SELECT * FROM STUDENT_INFO ORDER BY STUDENT_NAME DESC;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
27	Venkatesh	Chennai	24	Good	75
21	Veeramani	Madurai	21	Average	67
23	Veera	Tiruppur	20	Average	51
17	Swetha	Tiruchirapalli	17	Good	73
12	Suja	Salem	23	Very Good	85
15	Sindhu	Erode	22	Average	65
18	Selvi	Kumarapalayam	22	Average	47
28	Raja	Madurai	24	Average	42
19	Pooja	Salem	19	Very Good	88

BETWEEN:

THE BETWEEN OPERATOR SELECTS VALUES WITHIN A GIVEN RANGE. THE VALUES CAN BE NUMBERS, TEXT, OR DATES.

QUERY: SELECT * FROM STUDENT_INFO WHERE MARKS BETWEEN 70 AND 90;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
5	Moorthy	Salem	18	Very Good	86
7	Jaga	Madurai	24	Very Good	89
12	Suja	Salem	23	Very Good	85
16	Madhavi	Tiruppur	20	Good	78
17	Swetha	Tiruchirapalli	17	Good	73

GROUP BY:

THE ***GROUP*** BY STATEMENT ***GROUPS*** ROWS THAT HAVE THE SAME VALUES INTO SUMMARY ROWS, LIKE "FIND THE NP0]]UMBER OF CUSTOMERS IN EACH COUNTRY".

QUERY: SELECT CITY_STATE, COUNT(STUDENT_ID) FROM STUDENT_INFO GROUP BY CITY_STATE;

OUTPUT:

City_state	count(Student_id)
Chennai	4
Erode	4
Kumarapalayam	4
Madurai	4
Salem	4
Tiruchirapalli	4
Tiruppur	4

Limit: The LIMIT clause is used to specify the number of records to return.
QUERY: select * from Student_info order by Student_id Limit 3;
Output:

-

Student_id	Student_name	City_state	age	Results	marks
1	Geetha	Erode	21	No Rank	37
2	Guru	Tiruppur	20	No Rank	28
3	Gokul	Tiruchirapalli	18	Average	40
NULL	NULL	NULL	NULL	NULL	NULL

Desc Limit: the query returns a single record in the result set. In this case, the record with the maximum value in last_order_date .
QUERY: select * from Student_info order by Student_id desc Limit 0,4;
Output:

Student_id	Student_name	City_state	age	Results	marks
28	Raja	Madurai	24	Average	42
27	Venkatesh	Chennai	24	Good	75
26	Keerthi	Salem	17	Very Good	89
25	Devan	Kumarapalayam	21	Excellent	100
NULL	NULL	NULL	NULL	NULL	NULL

Greater THAN :

The greater (>) than operator is used to test whether an expression is greater than another one.

QUERY: select * from Student_info where age > 23;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
4	Mani	Kumarapalayam	24	No Rank	31
7	Jaga	Madurai	24	Very Good	89
10	Kabilan	Tiruchirapalli	24	Average	67
27	Venkatesh	Chennai	24	Good	75
28	Raja	Madurai	24	Average	42

GREATER THAN OR EQUAL TO :

The Greater than equal to (>=)shows the Higher values and also equal to that values

QUERY: select * from Student_info where marks >=70;

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
5	Moorthy	Salem	18	Very Good	86
7	Jaga	Madurai	24	Very Good	89
11	Manasi	Kumarapalayam	17	Excellent	97
12	Suja	Salem	23	Very Good	85
16	Madhavi	Tiruppur	20	Good	78

EQUAL TO:

the **equal** to operator is useful to check whether the given two expressions are equal or not. If it's equal, then the condition will be true, returning matched records.

QUERY: select * from Student_info where City_state = 'Salem';

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
5	Moorthy	Salem	18	Very Good	86
12	Suja	Salem	23	Very Good	85
19	Pooja	Salem	19	Very Good	88
26	Keerthi	Salem	17	Very Good	89
NULL	NULL	NULL	NULL	NULL	NULL

NOT EQUAL TO:

the **not equal** operator is used to check whether two expressions are equal or not. If it's not equal, then the condition will be true, and it will return not matched records.

QUERY: select * from Student_info where city_state != 'chennai';

OUTPUT:

Student_id	Student_name	City_state	age	Results	marks
1	Geetha	Erode	21	No Rank	37
2	Guru	Tiruppur	20	No Rank	28
3	Gokul	Tiruchirapalli	18	Average	40
4	Mani	Kumarapalayam	24	No Rank	31
5	Moorthy	Salem	18	Very Good	86

udent info 41 X

MY SQL CALCULATE FUNCTIONS

SUM:

SELECT SUM(MARKS) FROM STUDENT_INFO

	sum(marks)
▶	1865

AVERAGE

SELECT ROUND(AVG(MARKS), 0) FROM STUDENT_INFO;

	round(avg(marks),0)
▶	67

MIN

SELECT MIN(MARKS) FROM STUDENT_INFO;

	min(marks)
▶	28

MAX

SELECT MAX(MARKS) FROM STUDENT_INFO;

	max(marks)
▶	100

MYSQL STRING FUNCTIONS

LCASE -- THE LCASE

FUNCTION RETURNS A STRING IN WHICH ALL THE CHARACTERS ARE CONVERTED TO LOWERCASE CHARACTERS.

QUERY : SELECT LCASE(STUDENT_NAME) FROM STUDENT_DET;

OUTPUT

student_id	student_name	student_initial	marks	Gender	lcase(student_name)
14001	Guru	L	35	Male	guru
14002	Gopi	S	54	Male	gopi
14003	Sudhakar	D	78	Male	sudhakar

UCASE: THE UCASE

FUNCTION RETURNS A STRING IN WHICH ALL THE CHARACTERS HAVE BEEN CONVERTED TO UPPERCASE CHARACTERS,

QUERY :SELECT *,UCASE(STUDENT_NAME) FROM STUDENT_DET;

OUTPUT

student_id	student_name	student_initial	marks	Gender	ucase(student_name)
14001	Guru	L	35	Male	GURU
14002	Gopi	S	54	Male	GOPI
14003	Sudhakar	D	78	Male	SUDHAKAR

LEFT : THE ***LEFT*** () ***FUNCTION*** EXTRACTS A NUMBER OF CHARACTERS FROM A STRING (STARTING FROM ***LEFT***).

QUERY : SELECT LEFT(STUDENT_ID,2) AS DEF_ID, STUDENT_NAME, STUDENT_INITIAL, MARKS GENDER FROM STUDENT_DET;

Def_id	student_name	student_initial	marks	gender
14	Guru	L	35	Male
14	Gopi	S	54	Male
14	Sudhakar	D	78	Male

THE ***RIGHT*** () ***FUNCTION*** EXTRACTS A NUMBER OF CHARACTERS FROM A STRING (STARTING FROM ***RIGHT***).

QUERY : SELECT RIGHT(STUDENT_ID,3) AS DEF_ID, STUDENT_NAME, STUDENT_INITIAL, MARKS, GENDER FROM STUDENT_DET;

Def_id	student_name	student_initial	marks	Gender
001	Guru	L	35	Male
002	Gopi	S	54	Male
003	Sudhakar	D	78	Male

CONCAT: THE ***CONCAT*()** ***FUNCTION*** ADDS TWO OR MORE STRINGS TOGETHER.

QUERY: SELECT *,CONCAT(STUDENT_NAME, "" ,STUDENT_INITIAL)

AS STUDENT_FULL_NAME FROM STUDENT_DET;

OUTPUT

-

student_id	student_name	student_initial	marks	Gender	student_full_name
14001	Guru	L	35	Male	GuruL
14002	Gopi	S	54	Male	GopiS
14003	Sudhakar	D	78	Male	SudhakarD

TRIM: THE ***TRIM*()** ***FUNCTION*** REMOVES THE SPACE CHARACTER OR OTHER SPECIFIED CHARACTERS FROM THE START OR END OF A STRING.

QUERY: SELECT TRIM(STUDENT_NAME) FROM SUDENT_DET ;

OUTPUT

student_id	student_name	student_initial	marks	Gender	trim(Student_name)
14001	Guru	L	35	Male	Guru
14002	Gopi	S	54	Male	Gopi
14003	Sudhakar	D	78	Male	Sudhakar

CHAR_LENGTH

QUERY : SELECT * FROM STUDENT_DET WHERE CHAR_LENGTH(STUDENT_NAME)=4;

OUTPUT

student_id	student_name	student_initial	marks	Gender
14001	Guru	L	35	Male
14002	Gopi	S	54	Male
NULL	NULL	NULL	NULL	NULL

-

MID : THE ***MID*()** ***FUNCTION*** EXTRACTS SOME CHARACTERS FROM A ***STRING*** (STARTING AT ANY POSITION). SYNTAX. ***MID*(*STRING***, START, LENGTH).

QUERY: SELECT MID(STUDENT_ID,3,2)

AS MID_ID FROM STUDENT_DET;

OUTPUT

student_id	student_name	student_initial	marks	Gender	mid_id
14001	Guru	L	35	Male	00
14002	Gopi	S	54	Male	00
14003	Sudhakar	D	78	Male	00

MY SQL DATE FUNCTIONS

DATE ADD :

QUERY: SELECT *, DATE_ADD(HIRE_DATE,INTERVAL 5 MONTH) AS MONTH_ADD FROM EMP_DET;

OUTPUT:

emp_id	EMp_name	Job	Mgr	Hire_date	salary	commission	Dep_no	Month_add
7369	SMITH	CLERK	7902	1980-12-17	800	NULL	20	1981-05-17
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30	1981-07-20
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30	1981-07-22
7566	JONES	MANAGER	7839	1981-04-02	2975	NULL	20	1981-09-02
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	NULL	30	1982-02-28
7698	BLAKE	MANAGER	7839	1981-05-01	2850	1400	30	1981-10-01
7782	CLARK	MANAGER	7839	1981-06-09	2450	NULL	10	1981-11-09
7788	SCOTT	ANALYST	7566	1982-12-09	3000	NULL	20	1983-05-09
7839	KING	PRESIDENT	NULL	1981-11-17	5000	NULL	10	1982-04-17
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30	1982-02-08
7876	ADAMS	CLERK	7788	1983-01-12	1100	NULL	20	1983-06-12
7900	JAMES	CLERK	7698	1981-12-03	950	NULL	30	1982-05-03
7902	FORD	ANALYST	7566	1981-12-03	3000	NULL	20	1982-05-03

DATEDIFF:

QUERY: SELECT *, ROUND(DATEDIFF(CURDATE(),HIRE_DATE)/31,1)

AS EMP_EXP FROM EMP_DET;

OUTPUT:

emp_id	EMp_name	Job	Mgr	Hire_date	salary	commission	Dep_no	Emp_exp
7369	SMITH	CLERK	7902	1980-12-17	800	NULL	20	508.1
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30	506.0
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30	505.9
7566	JONES	MANAGER	7839	1981-04-02	2975	NULL	20	504.6
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	NULL	30	498.9
7698	BLAKE	MANAGER	7839	1981-05-01	2850	1400	30	503.7
7782	CLARK	MANAGER	7839	1981-06-09	2450	NULL	10	502.5
7788	SCOTT	ANALYST	7566	1982-12-09	3000	NULL	20	484.8
7839	KING	PRESIDENT	NULL	1981-11-17	5000	NULL	10	497.3
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30	499.5
7876	ADAMS	CLERK	7788	1983-01-12	1100	NULL	20	483.7
7900	JAMES	CLERK	7698	1981-12-03	950	NULL	30	496.7
7902	FORD	ANALYST	7566	1981-12-03	3000	NULL	20	496.7

TIMESTAMP DIFF

QUERY: SELECT *, TIMESTAMPDIFF(YEAR, HIRE_DATE, CURDATE()) AS EMP_EXP FROM

EMP_DET;

OUTPUT:

emp_id	EMp_name	Job	Mgr	Hire_date	salary	commission	Dep_no	emp_exp
7369	SMITH	CLERK	7902	1980-12-17	800	NULL	20	43
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30	42
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30	42
7566	JONES	MANAGER	7839	1981-04-02	2975	NULL	20	42
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	NULL	30	42
7698	BLAKE	MANAGER	7839	1981-05-01	2850	1400	30	42
7782	CLARK	MANAGER	7839	1981-06-09	2450	NULL	10	42
7788	SCOTT	ANALYST	7566	1982-12-09	3000	NULL	20	41
7839	KING	PRESIDENT	NULL	1981-11-17	5000	NULL	10	42
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30	42
7876	ADAMS	CLERK	7788	1983-01-12	1100	NULL	20	41
7900	JAMES	CLERK	7698	1981-12-03	950	NULL	30	42
7902	FORD	ANALYST	7566	1981-12-03	3000	NULL	20	42

DATE FORMAT

QUERY: SELECT *, DATE_FORMAT(HIRE_DATE, '%A') AS WEEK_NAME FROM EMP_DET;

emp_id	EMp_name	Job	Mgr	Hire_date	salary	commission	Dep_no	week_name
7369	SMITH	CLERK	7902	1980-12-17	800	NULL	20	Wed
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30	Fri
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30	Sun
7566	JONES	MANAGER	7839	1981-04-02	2975	NULL	20	Thu
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	NULL	30	Mon
7698	BLAKE	MANAGER	7839	1981-05-01	2850	1400	30	Fri
7782	CLARK	MANAGER	7839	1981-06-09	2450	NULL	10	Tue
7788	SCOTT	ANALYST	7566	1982-12-09	3000	NULL	20	Thu
7839	KING	PRESIDENT	NULL	1981-11-17	5000	NULL	10	Tue
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30	Tue
7876	ADAMS	CLERK	7788	1983-01-12	1100	NULL	20	Wed
7900	JAMES	CLERK	7698	1981-12-03	950	NULL	30	Thu

YEAR FORMAT

QUERY: SELECT * FROM EMP_DET WHE

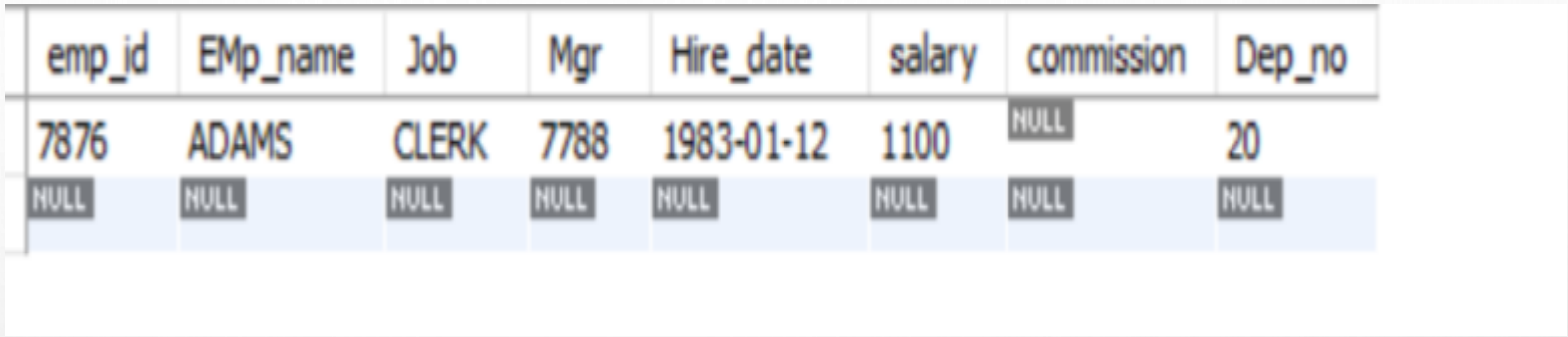
YEAR(HIRE_DATE)='1981';

emp_id	EMp_name	Job	Mgr	Hire_date	salary	commission	Dep_no
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30
7566	JONES	MANAGER	7839	1981-04-02	2975	NULL	20
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	NULL	30
7698	BLAKE	MANAGER	7839	1981-05-01	2850	1400	30
7782	CLARK	MANAGER	7839	1981-06-09	2450	NULL	10
7839	KING	PRESIDENT	NULL	1981-11-17	5000	NULL	10
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30
7900	JAMES	CLERK	7698	1981-12-03	950	NULL	30
7902	FORD	ANALYST	7566	1981-12-03	3000	NULL	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

DAY

QUERY: SELECT * FROM EMP_DET WHERE DAY (HIRE_DATE)='12';

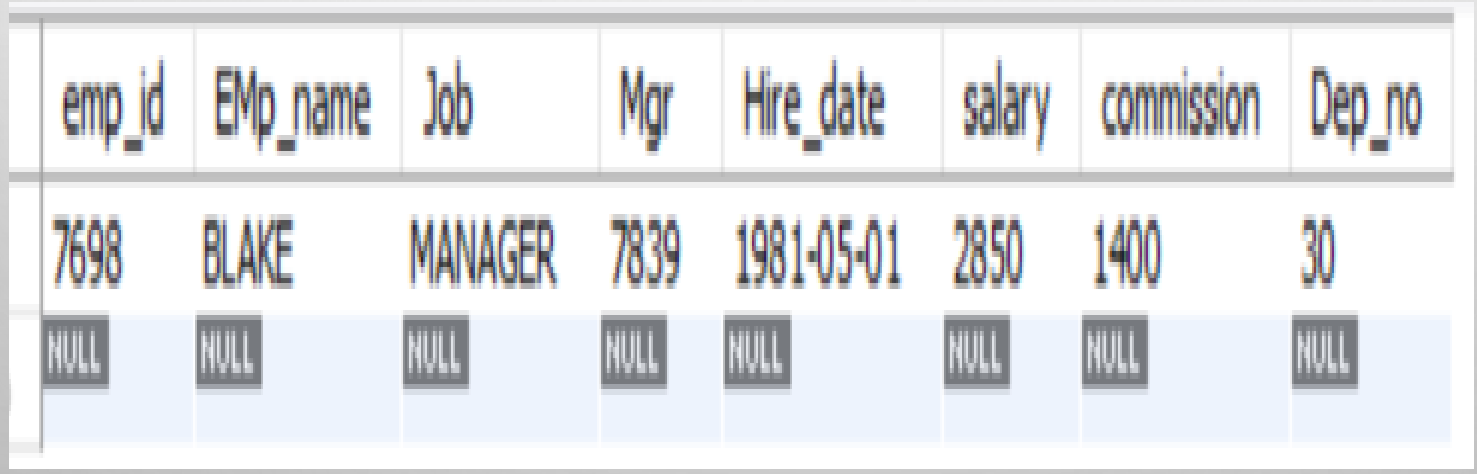
OUTPUT:



emp_id	EMp_name	Job	Mgr	Hire_date	salary	commission	Dep_no
7876	ADAMS	CLERK	7788	1983-01-12	1100	NULL	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- MONTH

QUERY: SELECT * FROM EMP_DET WHERE MONTH (HIRE_DATE)='05';



emp_id	EMp_name	Job	Mgr	Hire_date	salary	commission	Dep_no
7698	BLAKE	MANAGER	7839	1981-05-01	2850	1400	30
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

LOGICAL FUNCTIONS

IF

QUERY: SELECT STUDENT_NAME, IF(ATT_STATUS = ' P','1','0')
AS PRESENT_COUNT FROM ATT_INFO;

OUTPUT:

Student_name	Present_count
Vasanth	1
Vasanth	1
Vasanth	1
Vasanth	0
Guru	1
Guru	1
Guru	0
Guru	0

COUNT IF

QUERY: SELECT STUDENT_NAME, COUNT(IF(ATT_STATUS =' P','1',NULL))

AS IND_PRESENT_COUNT,

COUNT(IF(ATT_STATUS =' A','1',NULL))

AS IND_ABSENT_COUNT FROM ATT_INFO

GROUP BY STUDENT_NAME;

OUTPUT:

Student_name	Ind_present_count	Ind_Absent_count
Amutha	2	2
Arthi	1	3
Arun	3	1
Deepa	2	2
Devan	2	2
Devi	2	2
Gokul	1	3
Guru	2	2

IF WITH OR CONDITIONS

QUERY: SELECT *, IF(AVERAGE>=35,"PASS","FAIL")
AS RESULTS,
IF((TAMIL<=35) OR(ENGLISH<=35) OR
(ACCOUNTS<=35) OR (COMMERCE<=35) OR
(ECONOMICS <=35),"FAIL","PASS")
AS SUB_WISE_RESULTS
FROM MARKS_DET;

Roll_no	Student_name	Tamil	english	accounts	commerce	economics	Total	Average	Results	Sub_wise_results
100301	Govind	89	84	85	87	95	440	88	pass	Pass
100302	Priya	78	54	95	45	55	327	65	pass	Pass
100303	Ganesh	63	64	67	68	75	337	67	pass	Pass
100304	Keerthi	15	12	45	55	32	159	32	Fail	Fail
100305	Murali	88	92	94	97	91	462	92	pass	Pass
100306	Venkatesh	45	15	56	50	70	236	47	pass	Fail
100307	Bala	76	65	97	54	90	382	76	pass	Pass

IF WITH AND CONDITIONS

QUERY: SELECT *,IF(AMOUNT>='35000',"GOOD SALARY", "LOW SALARY") AS SAL_DET, IF ((AMOUNT>='30000') AND (AMOUNT>25000), "GOOD SALARY", "LOW SALARY") AS SUB_SAL_DET FROM SAL_DET;

OUTPUT:

Sal_id	emp_id	sal_date	amount	sal_det	sub_sal_det
121	1	2022-06-10	30000	low salary	Good salary
134	3	2022-06-15	23000	low salary	low salary
156	2	2022-06-12	18000	low salary	low salary
167	4	2022-06-20	35000	Good salary	Good salary
178	5	2022-06-23	20000	low salary	low salary

Join Queries:

A JOIN clause is used to combine rows from two or more tables, based on a related column between them

Inner Join : The ***INNER JOIN*** keyword selects records that have matching values in both tables.
Query: select * from emp_data inner join sal_det on emp_data.emp_id=sal_det.emp_id;

emp_id	emp_name	Designation	Date_of_join	Sal_id	emp_id	sal_date	amount
1	Guru	Manager	2022-05-10	121	1	2022-06-10	10000
3	Mani	Senior Manager	2022-05-15	134	3	2022-06-15	12000
2	Gopi	Junior Accountant	2022-05-12	156	2	2022-06-12	18000
4	Moorthy	HR	2022-05-20	167	4	2022-06-20	16000
5	Muthu	General Manager	2022-05-23	178	5	2022-06-23	12000

Cross join: The ***CROSS JOIN*** keyword returns all records from both tables (table1 and table2).
QUERY :select * from emp_data cross join sal_det;

emp_id	emp_name	Designation	Date_of_join	Sal_id	emp_id	sal_date	amount
1	Guru	Manager	2022-05-10	121	1	2022-06-10	10000
1	Guru	Manager	2022-05-10	134	3	2022-06-15	12000
1	Guru	Manager	2022-05-10	156	2	2022-06-12	18000
1	Guru	Manager	2022-05-10	167	4	2022-06-20	16000
1	Guru	Manager	2022-05-10	178	5	2022-06-23	12000
2	Gopi	Junior Accountant	2022-05-12	121	1	2022-06-10	10000
2	Gopi	Junior Accountant	2022-05-12	134	3	2022-06-15	12000
2	Gopi	Junior Accountant	2022-05-12	156	2	2022-06-12	18000
2	Gopi	Junior Accountant	2022-05-12	167	4	2022-06-20	16000
2	Gopi	Junior Accountant	2022-05-12	178	5	2022-06-23	12000
3	Mani	Senior Manager	2022-05-15	121	1	2022-06-10	10000

Left Join: The ***LEFT JOIN*** keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).
QUERY:select * from emp_data left join sal_det on emp_data.emp_id=sal_det.emp_id;

emp_id	emp_name	Designation	Date_of_join	Sal_id	emp_id	sal_date	amount
1	Guru	Manager	2022-05-10	121	1	2022-06-10	10000
3	Mani	Senior Manager	2022-05-15	134	3	2022-06-15	12000
2	Gopi	Junior Accountant	2022-05-12	156	2	2022-06-12	18000
4	Moorthy	HR	2022-05-20	167	4	2022-06-20	16000
5	Muthu	General Manager	2022-05-23	178	5	2022-06-23	12000
6	Abhi	Associate	2022-06-05	NULL	NULL	NULL	NULL

Right Join: The ***RIGHT JOIN*** keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1)
QUERY: select * from emp_data right join sal_det on emp_data.emp_id = sal_det.emp_id;

emp_id	emp_name	Designation	Date_of_join	Sal_id	emp_id	sal_date	amount
1	Guru	Manager	2022-05-10	121	1	2022-06-10	10000
3	Mani	Senior Manager	2022-05-15	134	3	2022-06-15	12000
2	Gopi	Junior Accountant	2022-05-12	156	2	2022-06-12	18000
4	Moorthy	HR	2022-05-20	167	4	2022-06-20	16000
5	Muthu	General Manager	2022-05-23	178	5	2022-06-23	12000

Full Outer Join: The ***FULL OUTER JOIN*** keyword returns all records when there is a match in left (table1) or right (table2) table records.

QUERY :(select * from emp_data left join sal_det on emp_data.emp_id = sal_det.emp_id);

emp_id	emp_name	Designation	Date_of_join	Sal_id	emp_id	sal_date	amount
1	Guru	Manager	2022-05-10	121	1	2022-06-10	10000
3	Mani	Senior Manager	2022-05-15	134	3	2022-06-15	12000
2	Gopi	Junior Accountant	2022-05-12	156	2	2022-06-12	18000
4	Moorthy	HR	2022-05-20	167	4	2022-06-20	16000
5	Muthu	General Manager	2022-05-23	178	5	2022-06-23	12000
6	Abhi	Associate	2022-06-05	NULL	NULL	NULL	NULL

Union: The **UNION** operator is used to combine the result-set of two or more

QUERY:(select * from emp_data left join sal_det onemp_data.emp_id = sal_det.emp_id)union(select * from emp_data right join sal_det onemp_data.emp_id = sal_det.emp_id);

emp_id	emp_name	Designation	Date_of_join	Sal_id	emp_id	sal_date	amount
1	Guru	Manager	2022-05-10	121	1	2022-06-10	10000
3	Mani	Senior Manager	2022-05-15	134	3	2022-06-15	12000
2	Gopi	Junior Accountant	2022-05-12	156	2	2022-06-12	18000
4	Moorthy	HR	2022-05-20	167	4	2022-06-20	16000
5	Muthu	General Manager	2022-05-23	178	5	2022-06-23	12000
6	Abhi	Associate	2022-06-05	NULL	NULL	NULL	NULL

STORED PROCEDURE

A STORED PROCEDURE IS A PREPARED SQL CODE THAT YOU CAN SAVE, SO THE CODE CAN BE REUSED OVER AND OVER AGAIN

- QUERY:

```
DELIMITER //
CREATE PROCEDURE STORE_DATA7()
BEGIN
 SELECT * FROM STUDENT_DET WHERE CITY_STATE ='CHENNAI' OR CITY_STATE = 'MADURAI';
SELECT * FROM STUDENT_DET WHERE AGE>=22;
SELECT * FROM STUDENT_DET WHERE RESULT = GOOD;
```

```
SELECT *,
 CASE
WHEN CITY_STATE = 'ERODE' THEN 500
WHEN CITY_STATE = 'TIRUPPUR' THEN 700
WHEN CITY_STATE = 'TIRUCHIRAPALLI' THEN 400
WHEN CITY_STATE = 'KUMARAPALAYAM' THEN 350
WHEN CITY_STATE = 'SALEM' THEN 700
WHEN CITY_STATE = 'CHENNAI' THEN 700
WHEN CITY_STATE = 'SALEM' THEN 700
 ELSE 'NO DATA'
END AS BUS_PASS_AMOUNT
FROM STUDENT_DET;
 END //
DELIMITER ;
CALL STORE_DATA7;
```

student_id	student_name	city_state	age	Result	marks
6	Amutha	Chennai	17	Average	61
7	Jaga	Madurai	24	VeryGood	89
13	Arun	Chennai	22	NoRank	32
14	Deepa	Madurai	20	Average	49

student_id	student_name	city_state	age	Result	marks
4	Mani	Kumarapalayam	24	NoRank	31
7	Jaga	Madurai	24	VeryGood	89
8	Pavithra	Erode	23	Average	68
10	Kabilan	Tiruchirapalli	24	Average	67
12	Suria	Salem	23	VeryGood	85

TRIGGER:

A **database trigger is stored program which is automatically fired or executed when some events occur**

Types of Trigger:-

- Row level trigger – An event is triggered at row level for each row updated, inserted or deleted.
- Statement level trigger- An event is triggered at table level for each Sql statement executed
- .

***Trigger timings*:-**

- After insert
- Before insert
- Before update
- After update
- Before delete
- After delete

ROW LEVEL TRIGGER

BEFORE INSERT

```
CREATE TABLE STUDENT_DETAIL1 (STUDENT_ID INT, STUDENT_NAME VARCHAR(25), CITY_STATE
 VARCHAR(30),AGE INT, COMMUNITY VARCHAR(150), MARKS INT, PRIMARY KEY(STUDENT_ID))
DELIMITER //
 CREATE TRIGGER AGE_CHECK
BEFORE INSERT ON STUDENT_DETAIL1 FOR EACH ROW
BEGIN
IF NEW.AGE <0 THEN
SET NEW.AGE = 0;
END IF;
END //
DELIMITER ;
 INSERT INTO STUDENT_DETAIL1 VALUES
(3,          'GOKUL',          'TIRUCHIRAPALLI',          18,          'BC',          89),
(4,          'MANI',   'KUMARAPALAYAM',          -24,          'BC',          56);
 SELECT * FROM STUDENT_DETAIL1;
```

```
DELIMITER //
CREATE TRIGGER COMMUNITY_CHECK BEFORE INSERT ON STUDENT_DETAIL1 FOR EACH
ROW
BEGIN
IF NEW.COMMUNITY IS NULL
THEN SET NEW.COMMUNITY= 'KINDLY UPDATE YOUR COMMUNITY';
 END IF;
 END //
 DELIMITER ;
 INSERT INTO STUDENT_DETAIL1 VALUES
(5, 'MANI', 'KUMARAPALAYAM', 24, 'BC', 56),
(6, 'MOORTHY', 'SALEM', -18, NULL, 90);
SELECT * FROM STUDENT_DETAIL1;
```

OUTPUT:

Student_id	student_name	city_state	age	community	marks
3	Gokul	Tiruchirapalli	18	bc	89
4	Mani	Kumarapalayam	0	BC	56
5	mani	Kumarapalayam	24	Bc	56
6	Moorthy	salem	0	Kindly update your community	90
NULL	NULL	NULL	NULL	NULL	NULL

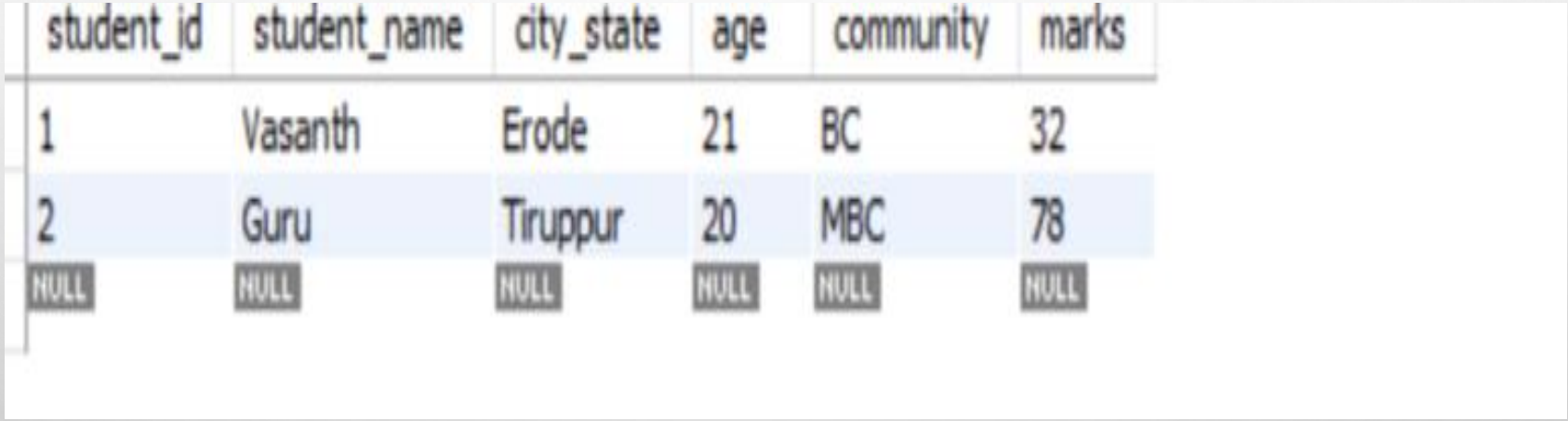
STATEMENT LEVEL TRIGGER

AFTER INSERT:

```
CREATE TABLE STUDENT_DET (STUDENT_ID INT, STUDENT_NAME VARCHAR(25),
CITY_STATE VARCHAR(25), AGE INT, COMMUNITY VARCHAR(30), MARKS INT, PRIMARY
KEY(STUDENT_ID));
CREATE TABLE STUDENT_BACKUP_DET (STUDENT_ID INT, STUDENT_NAME VARCHAR(25),
CITY_STATE VARCHAR(25), AGE INT, COMMUNITY VARCHAR(30),MARKS INT,
 PRIMARY KEY(STUDENT_ID));
DELIMITER //
CREATE TRIGGER BACKUP_UPDATE AFTER INSERT ON STUDENT_DET FOR EACH ROW
BEGIN
INSERT INTO STUDENT_BACKUP_DET(STUDENT_ID, STUDENT_NAME, CITY_STATE, AGE,
COMMUNITY, MARKS)
VALUES (NEW.STUDENT_ID, NEW. STUDENT_NAME, NEW.CITY_STATE,
NEW. AGE, NEW. COMMUNITY, NEW.MARKS);
END //
DELIMITER ;
```

```
INSERT INTO STUDENT_DET VALUES
(1,      'VASANTH',      'ERODE',21,      'BC',      32),
(2,      'GURU',  'TIRUPPUR',      20,      'MBC',    78);
SELECT * FROM STUDENT_DET;
SELECT * FROM STUDENT_BACKUP_DET;
```

OUTPUT:



student_id	student_name	city_state	age	community	marks
1	Vasanth	Erode	21	BC	32
2	Guru	Tiruppur	20	MBC	78
NULL	NULL	NULL	NULL	NULL	NULL

THANK YOU