SQL

Creating new database 🡪

CREATE DATABASE mybd // It will throw error if database is already created

CREATE DATABASE IF NOT EXISTS mybd // will check and then create database

Types of SQL Commands 🡪

* DDL Commands (Data Definition Language)
  + CREATE
  + ALTER
  + DROP
  + TRUNCATE
  + RENAME
* DML Commands (Data Manipulation Language)
  + INSERT
  + UPDATE
  + DELETE
* DRL / DQL Commands (Data Retrieval / Data Query Language)
  + SELECT
* TCL Commands (Transaction Control Language)
  + COMMIT
  + ROLLBACK
  + SAVE POINT
* DCL Commands (Data Control Language)
  + GRAND
  + REVOKE

**Creating a Table 🡪**

CREATE TABLE STUDENT (

SNO INT(5),

SNAME VARCHAR(15),

MARKS INT(3),

GRADE CHAR(2)

);

To get info about table like column names, data types etc,

DESCRIBE STUDENT;

Adding data to the values 🡪

INSERT INTO STUDENT VALUES(101, ‘VIJAY’, 84, ‘AA’) // It will accept the values in column sequence

INSERT INTO STUDENT(SNAME, SNO, MARKS, GRADE) VALUES(‘VIJAY’, 101, 84, ‘AA’) // It will accept the values in specified order

INSERT INTO STUDENT VALUES(101, ‘VIJAY’, NULL, ‘AA’) // If you don’t know any value

**DDL Commands (Data Definition Language)**

* CREATE
* ALTER
* DROP
* TRUNCATE
* RENAME

ALTER

* Adding a new column

ALTER TABLE STUDENT ADD(GRADE VARCHAR(2))

* Dropping a column from table

ALTER TABLE STUDENT DROP COLUMN GRADE

* Modify existing column (using constrains)

ALTER TABLE STUDENT MODIFY COLUMN SNAME VARCHAR(20)

* Rename the column

ALTER TABLE STUDENT RENAME COLUMN SNAME TO STUNAME

* Rename the table

ALTER TABLE STUDENT TO GOOD\_BOYS

DROP 🡪 It will delete table, data everything from the database.

DROP TABLE STUDENT

TRUNCATE 🡪 It will delete the data but table will be there. Data will be removed permanently.

TRUNCAE TABLE STUDENT

DELETE 🡪 It is same as TRUNCATE except data will be removed temporarily. Here we can rollback the data if we want.

DELETE FROM STUDENT

Alias in SQL 🡪

Alias are the names that we want to see to the column after executing the query.

SELECT EMPLOYEE\_ID EMPID, FIRST\_NAME NAME, SALARY SAL FROM EMPLOYEE

Here, EMPID is alias for EMPLOYEE\_ID and NAME is alias for FIRST\_NAME.

**WHERE Clause**

* It is used to filter the data based on condition.

We can’t apply arithmetic operators (like ‘=’) on NULL keyword. You have to give words like ‘is’.

SELECT \* FROM EMPLOYEE WHERE ADDRESS IS NULL

DISTINCT 🡪 It is used to get unique data.

SELECT DISTINCT DEPT\_ID FROM EMPLOYEE

Logical Operators 🡪 AND, OR, NOT

SELECT \* FROM EMPLOYEE WHERE SALARY>20000 AND NAME=’AJAY’

SELECT NAME FROM EMPLOYEE WHERE DEPT\_ID=’HR’ OR SALARY>50000

SELECT \* FROM EMPLOYEE WHERE NOT NAME=’VJ’

BETWEEN 🡪

SELECT \* FROM EMPLOYEE WHERE SALALRY BETWEEN 20000 AND 30000, it can be written as below

SELECT \* FROM EMPLOYEE WHERE SALARY>20000 AND SALARY<30000

NOT BETWEEN 🡪

SELECT \* FROM EMPLOYEE WHERE SALARY NOT BETWEEN 70000 AND 90000

IN 🡪

SELECT \* FROM EMPLOYEE WHERE SALARY IN(10000, 20000, 30000), it can also be written as below

SELECT \* FROM EMPLOYEE WHERE SALARY = 10000 OR SALARY = 20000 OR SALARY = 30000

NOT IN 🡪

SELECT \* FROM EMPLOYEE WHERE SALARY NOT IN(10000, 20000, 30000)

Patten Matching Operator 🡪

% 🡪 many characters

\_ 🡪 single character

We need to use LIKE keyword in pattern matching, we can’t use ‘=’ operator.

Find the data of employee whose first name starts with ‘V’

SELECT \* FROM EMPLOYEE WHERE NAME LIKE ‘V%’

Find the data of employee whose first name starts with ‘V’ and ends with ‘T’

SELECT \* FROM EMPLOYEE WHERE NAME LIKE ‘V%T’

Find the data of employee whose first name has second last alphabet as ‘J’

SELECT \* FROM EMPLOYEE WHERE NAME LIKE ‘%J\_’

Find the data of employee whose first name has only three alphabets

SELECT \* FROM EMPLOYEE WHERE NAME LIKE ‘\_\_\_’

Find the data of employee whose first name do not starts with ‘A’

SELECT \* FROM EMPLOYEE WHERE NAME NOT LIKE ‘A%’

**Built-in Functions**

String Functions 🡪

* UPPER – Returns upper case word. It wont change in table but just present it by converting

SELECT UPPER (‘mine’)

* LOWER
* LENGTH – Returns length of characters

SELECT LENGTH (‘welcome’)

SELECT LENGTH (FIRST\_NAME) FROM EMPLOYEE

Find list of employees whose name has four characters

SELECT \* FROM EMPLOYEE WHERE LENGTH(FIRST\_NAME)=4;

* TRIM – Removes white spaces

SELECT TRIM (‘ RAHUL ’)

SELECT TRIM (‘Z’ FROM ‘ ZZORACLEZZ ’) // will return ORACLE

* INSTR – returns the position of a character in string

SELECT INSTR (‘WELCOME’, ‘O’) // will return 4

* SUBSTR / SUBSTRING-

SELECT SUBSTR (‘ORACLE’, 2, 3) // will return RAC

SELECT SUBSTRING (FIRST\_NAME, 1, 3) FROM EMPLOYEE

* CONCAT – will join two strings

SELECT CONCAT (‘VJ’, ‘HERO’) // will return VJHERO

SELECT CONCAT(FIRST\_NAME, LAST\_NAME) FROM EMPLOYEE

For more details - <https://dev.mysql.com/doc/refman/8.0/en/string-functions.html>

Numeric Functions 🡪

* ABS – Absolute value SELECT ABS (-40)
* SQRT – Square root SELECT SQRT (25)
* MOD – returns remainder SELECT MOD (10,3)
* POWER - SELECT POWER(2,5) // will return 32
* TRUNCATE – Truncates the number to specified decimals

SELECT TRUNCATE (4.1234, 2) // will return 4.12

SELECT TRUNCATE (635891, -3) // will return 635000

* GREATEST

SELECT GREATEST (100,200,300) // will return 300

* LEAST

SELECT LEAST (100,200,300) // will return 100

Date Functions 🡪

* CURDATE / CURRENT\_DATE

SELECT CURDATE() // will return current today’s date

SELECT CURRENT\_DATE()

* CURTIME / CURRENT\_TIME

SELECT CURTIME()

SELECT CURRENT\_TIME()

* NOW

SELECT NOW() // will return current date and time

* YEAR – Extracts the year from provided date

SELECT YEAR (“2019-05-19”) // will return 2019

* MONTH – Extracts the month from provided date

SELECT MONTH (“2019-05-19”) // will return 05

* DAY – Extracts the date from provided date

SELECT DAY (“2019-05-19”) // will return 19

Print the data of employees who joined in 1987

SELECT \* FROM EMPLOYEE WHERE YEAR (HIRE\_DATE)= “1987”

Aggregate Functions 🡪

* AVG – Returns average

SELECT AVG(SALARY) FROM EMPLOYEE;

* SUM

SELECT SUM(SALARY) FROM EMPLOYEE;

* MIN

SELECT MIN(SALARY) FROM EMPLOYEE;

* MAX

SELECT MAX(SALARY) FROM EMPLOYEE;

* COUNT

SELECT COUNT(\*) FROM EMPLOYEE;

**GROUP BY, HAVING & ORDER BY Clauses**

GROUP BY 🡪 Grouping the data from result based on certain condition

Find sum of salaries of employees department wise

SELECT DEPART\_ID, SUM(SALARY) FROM EMPLOYEE GROUP BY DEPART\_ID

Find the average salary of all the departments

SELECT DEPART\_ID, AVG(SALARY) FROM EMPLOYEE GROUP BY DEPART\_ID

HAVING 🡪 It is used to put certain condition in grouped data

Find the department and its total count if that department has more than 20 people

SELECT DEPART\_ID, COUNT(\*) FROM EMPLOYEE GROUP BY DEPART\_ID HAVING COUNT(\*)>20

Find the sum of salaries of employees department wise if that department exceeds total salary of 2000

SELECT DEPART\_ID, SUM(SALARY) FROM EMPLOYEE GROUP BY DEPART\_ID HAVING SUM(SALARY)>2000

Difference between WHERE and HAVING class is that HAVING class is used on top of GROUP BY class for filtering the data but WHERE class is used in general to filter the data.

Find the sum of salaries of employees department wise if that department exceeds total salary of 2000 excluding a specific department (say 4)

SELECT DEPART\_ID, SUM(SALARY) FROM EMPLOYEE WHERE DEPART\_ID <> 4 GROUP BY DEPART\_ID HAVING SUM(SALARY)>2000;

ORDER BY 🡪

SELECT \* FROM EMPLOYEE ORDER BY SALARY // will retrieve data in ascending order based on Salary

SELECT \* FROM EMPLOYEE ORDER BY SALARY DESC // descending order

Order of Execution 🡪

WHERE -> GROUP BY -> HAVING -> ORDER BY

SELECT column\_name

WHERE condition

GROUP BY column\_name

HAVING condition

ORDER\_BY column\_name

Find the sum of salaries department wise if department’s salary exceeds 200000 in ascending order

SELECT DEPT\_ID, SUM(SALARY) FROM EMPLOYEE GROUP BY DEPT\_ID HAVING SUM (SALARY)>200000 ORDER BY SUM(SALARY)

Find the sum of salaries department wise (excluding DEPT\_ID=102) if department’s salary exceeds 200000 in descending order

SELECT DEPT\_ID, SUM(SALARY) FROM EMPLOYEE WHERE DEPT\_ID<>102 GROUP BY DEPT\_ID HAVING SUM (SALARY)>200000 ORDER BY SUM(SALARY) DESC

**Auto Increment**

* This property is defined during creating a table.
* Column for which we have to use AUTO\_INCREMENT, that column must be Primary Key.

CREATE TABLE STUDENT (

SNO INT(5) PRIMARY KEY, AUTO\_INCREMENT,

SNAME VARCHAR(15),

MARKS INT(3)

);

INSERT INTO STUDENT(SNAME, MARKS) VALUES(‘VIJAY’, 84); // (1, VIJAY, 84) will get inserted

INSERT INTO STUDENT(SNAME, MARKS) VALUES(‘ASHISH’, 40); // (2, ASHISH, 40) will get inserted

**LIMIT**

* Limit keyword is used to limit the number of rows that we are retrieving.

SELECT \* FROM EMPLOYEE LIMIT 5; // will retrieve first five rows

SELECT \* FROM EMPLOYEE LIMIT 5, 10; // will retrieve 10 rows starting from fifth rows

**VIEW**

* View is a virtual table based on the result set of an SQL statement.
* It is generally used if some other user wants access to our table. In such situation we create view of our table and give permission for access to other user. So any changes (insert, delete, modify) in it will not affect our original data.
* Remember View is just a representation, it will not create new table and will not consume new memory.
* Its syntax is CREATE VIEW ‘VIEW\_NAME’ AS followed by SQL query.

CREATE VIEW EMPLYOYEE\_V1 AS SELECT ID, NAME, SALARY FROM EMPLYOEE;

Now other user can access this new view EMPLYOYEE\_V1 as a table and regular table retrieve commands will work on it.

* Note : Any changes done in original table will affect the view but any changes done in view will not affect the table.

**INDEX**

* It is used to improve the performance, used to retrieve the data from the database very fast.
* In a table we can create index on one column or multiple columns.
* Generally the column which has complicated data, we create index on that column.
* User can’t see the indexes but can feel the increase in speed.

CREATE INDEX IDX\_NAME ON EMPLOYEES (SNAME);

* IDX\_NAME is name of the created index and SNAME is the column on which index is created.

DROP INDEX IDX\_NAME ON EMPLOYEES // to delete the index

**Joins**

* Join is used to get the data from multiple tables.
* For join to apply, there must be at least one common column between two or more tables.
* Types of Joins 🡪
  + Equi Join / Inner Join / Simple Join
  + Right Join
  + Left Join
  + Full Join
  + Self Join
* Inner Join 🡪 It will retrieve the common data from both the tables.

SELECT \* FROM TAB1 INNER JOIN TAB2 ON TAB1.NUMID = TAB2.NUMID

(you can read it as SELECT \* FROM TAB1 INNER JOIN TAB2; ON TAB1.NUMID = TAB2.NUMID)

* Left Join 🡪 It will retrieve matched data from both tables + unmatched data from left table, means it will retrieve complete left table.

SELECT \* FROM TAB1 LEFT JOIN TAB2 ON TAB1.NUMID = TAB2.NUMID

* Right Join 🡪 It will retrieve matched data from both tables + unmatched data from right table, means it will retrieve complete right table.

SELECT \* FROM TAB1 RIGHT JOIN TAB2 ON TAB1.NUMID = TAB2.NUMID

* Full Join 🡪 It will retrieve matched data from both tables + unmatched data from left table + unmatched data from right table means it will retrieve both the tables.

SELECT \* FROM TAB1 FULL JOIN TAB2 ON TAB1.NUMID = TAB2.NUMID

* Self Join 🡪 It will retrieve data from same table by satisfying specific condition (say a table has two columns Emp\_ID & his Manager\_ID and user want to retrieve data of only those employees who are managers)

SELECT E.EMPLOYEE\_ID, M.MANAGER\_ID FROM EMPLOYEE E, EMPLOYEE M WHERE E.MANAGER\_ID=M.EMPLOYEE\_ID // here E and M are aliases

**Sub-Query**

* A query within a query is called as ‘sub query’. Sub query should be written in brackets () so that these will be evaluated first.
* If sub query is returning single value then it is called as ‘single row sub query’. Here we can use <=, >=, !=
* If sub query is returning multiple values then it is called as ‘multi row sub query’. Here we can use IN, ANY, ALL

Example 1

* Suppose I want to find name of employees whose salary is grater than that of Vinit. Suppose salary of Vinit is 50000. So query will be,

SELECT EMP\_NAME FROM EMPLOYEE WHERE SALARY>50000

* But here firstly we need to write sub query to find the salary of Vinit.

SELECT SALARY FROM EMPLOYEE WHERE EMP\_NAME=’Vinit’

* So complete query will be

SELECT EMP\_NAME FROM EMPLOYEE WHERE SALARY>( SELECT SALARY FROM EMPLOYEE WHERE EMP\_NAME=’Vinit’)

Example 2

* Find the second highest salary from employees.
* Query to find the highest salary is

SELECT MAX(SALARY) FROM EMPLOYEE

* To find the second highest salary,

SELECT MAX(SALARY) FROM EMPLOYEE WHERE SALARY < (SELECT MAX(SALARY) FROM EMPLOYEE )

Example 3

* Find employee details those are having highest salary

SELECT EMP\_NAME FROM EMPLOYEE WHERE SALARY=(SELECT MAX(SALARY) FROM EMPLOYEE)

Example 4 - multi row sub query

* Display employee details whose salary is equals to the at least one employee of HR department

SELECT \* FROM EMPLOYEE WHERE SALARY IN (SELECT SALARY FROM EMPLOYEE WHERE DEPARTMENT = HR)

* Display employee details whose salary is greater than any employee of HR department

SELECT \* FROM EMPLOYEE WHERE SALARY > ANY (SELECT SALARY FROM EMPLOYEE WHERE DEPARTMENT = HR)

* Display employee details whose salary is less than all the employee of HR department

SELECT \* FROM EMPLOYEE WHERE SALARY < ALL (SELECT SALARY FROM EMPLOYEE WHERE DEPARTMENT = HR)

Example 5

* Display employees details whose salary is less than the maximum salary but hired before than the employee with higher salary

SELECT \* FROM EMPLOYEE WHERE SALARY < (SELECT MAX(SALARY) FROM EMPLOYEE)

AND HIRE\_DATE > (SELECT HIRE\_DATE FROM EMPLOYEE WHERE SALARY=

(SELECT MAX(SALARY) FROM EMPLOYEE))

**Integrity Constraints**

* Integrity constrains are used to specify rules for the data in the table.
* Constrains can be specified while creating the data using CREATE TABLE or after the table is created using ALTER TABLE statement.
* Constrains are always applied on columns and not on table.

Constrains Type 🡪

* + NOT NULL
  + UNIQUE
  + PRIMARY KEY
  + FOREIGN KEY
  + CHECK
  + DEFAULT

NOT NULL 🡪 will not accept null value

CREATE TABLE EMPLOYEE

(

SR\_NO INT(3) NOT NULL,

EMP\_NAME VARCHAR(25)

)

INSERT INTO EMPLOYEE VALUES (NULL, “VIJAY”) // It will throw an error

UNIQUE 🡪 will not accept any duplicate values

* It will allow multiple NULL value

CREATE TABLE EMPLOYEE

(

SR\_NO INT(3),

EMP\_NAME VARCHAR(25),

UNIQUE (SR\_NO) // Note – both ways of defining constrains (above and this) are same

)

PRIMARY KEY 🡪 It is combination of UNIQUE + NOT NULL

* Primary key column can be only one column in complete table

CREATE TABLE EMPLOYEE

(

SR\_NO INT(3) PRIMARY KEY,

EMP\_NAME VARCHAR(25)

)

* When we combine two columns to create a primary key (as no column is unique for that specific table), it is called composite key. It needs to be applied on table level and not on column level.

CREATE TABLE EMPLOYEE

(

SR\_NO INT(3),

EMP\_NAME VARCHAR(25),

AGE INT(2),

ADDRESS VARCHAR(25),

PRIMARY KEY (EMP\_NAME, AGE)

)

FOREIGN KEY 🡪

* It is used to link two tables.
* It is a field in one column that refers to the primary key in another table.
* Table containing foreign key is called as child table and the table containing primary key is called as referenced or parent table.

CREATE TABLE SCHOOL

(

SR INT(3),

NAME VARCHAR(15),

MARKS INT(3),

PRIMARY KEY (SR)

);

INSERT INTO SCHOOL VALUES(101, ‘VJ’, 99);

INSERT INTO SCHOOL VALUES(102, ‘Kiran’, 70);

INSERT INTO SCHOOL VALUES(103, ‘Vinit’, 40);

CREATE TABLE LIBRARY

(

SR INT(3),

BOOK\_NAME VARCHAR(15),

FOREIGN KEY(SR) REFERENCES SCHOOL(SR)

)

INSERT INTO LIBRARY VALUES (102, ‘Java’);

INSERT INTO LIBRARY VALUES (105, ‘Python’); // will throw error as 105 is not present in SR column of School table

* Here foreign key and reference column are need not to have same name but they must have same data type.
* If user tries to delete a record from parent table (whose dependency is there in child class), then it will throw an error unless corresponding record from child tables gets deleted first.

ON DELETE CASCADE 🡪

* Though there are some dependencies from child table, we can remove corresponding record from parent table using this option. After deleting record from parent table, automatically corresponding child table record will get deleted.
* We need to write this keyword while defining the child data itself and then delete the record from parent table record as usual.

CREATE TABLE LIBRARY

(

SR INT(3),

BOOK\_NAME VARCHAR(15),

FOREIGN KEY(SR) REFERENCES SCHOOL(SR) ON DELETE CASCADE

);

CHECK 🡪 It will allow to enter value after meeting certain checks.

CREATE TABLE STUDENT

(

SR INT(3),

NAME VARCHAR (15),

MARKS INT(3) CHECK(MARKS BETWEEN 50 AND 100)

);

Default Constraint 🡪 It is used if user do not provide any value then this will be taken as default.

CREATE TABLE ORDERS

(

ID INT(3),

ORDER\_NUMBER INT(5),

ORDER\_DATE DATETIME DEFAULT NOW() //NOW() is build inn function, DATETIME is data type

);

INSERT INTO ORDERS (ID, ORDER\_NUMBER) VALUES (101, 10231);

UNION & UNION ALL

* In UNION, tables must have same number of columns, same data type and should be in same order.
* UNION will fetch the data from both the tables (excluding duplicates) while UNION ALL will retrieve the data with duplicate values also.

SELECT NUM FROM A UNION SELECT NUM FROM B

SELECT NUM FROM A UNION ALL SELECT NUM FROM B