Agenda

```
DBMS VS RDBMS
SQL
SQL Categories
MySQL
Getting Started with MySQL
Database- Logical & Physical Layout
SQL Scripts
```

DBMS vs RDBMS

```
- DBMS -> File based System
- RDBMS -> Client Server based
    - Relational DataBase Management System
```

SQL

```
- Structured Query Language
- RQBE -> Relational Query By Example
- ANSI standarized in 1987 to SQL
- SQL is case insensitive
```

SQL categories

```
1. DDL -> Data Defination Language
    - CREATE,ALTER,DROP,RENAME,TRUNCATE
2. DML -> Data Manipulation Language
    - INSERT,UPDATE,DELETE
3. DQL -> Data Query Language
    - SELECT
4. DCL -> Data Control Language
    - CREATE USER,GRANT,REVOKE
5. TCL -> Transaction Control Language
    - SAVEPOINT,COMMIT,ROLLBACK
```

MySQL

```
MySQL was developed by Micheal Wideneus in 1995.
Named after his daughter 'Myia'
```

```
Sun Microsystems took over Mysql in 2008.
Oracle took over sunmicrosystems in 2010.
Mysql is free and open source under GPL.
```

Getting Started.

```
mysql -u root -p
    -u -> username
    -p -> password
-If you get an error saying can't connect to Mysql check if server is RUNNING OR
STOPPED
- mysql is not recogniozed command then set the path for mysql in environment
variables.
- Mysql gets installed in your machine at below locations
C:\Program Files\MySQL\MySQL Server 8.0
C:\Program Data\MySQL\MySQL Server 8.0
- C:\Program Files\MySQL\MySQL Server 8.0\bin ->Copy this path and paste it in
your environment variables.
```

Database

```
    mysql -u root -p

    SELECT DATABASE(); -- Null

    SHOW DATABASES; -- list of all exixting databases

    CREATE DATABASE sunbeam_students; --to create a new database

    SHOW DATABASES; -- you found your created db

    SELECT DATABASE(); -- NULL

    USE sunbeam_students; --To select the database for working

    SELECT DATABASE(); -- sunbeam_students
```

TABLES

```
    CREATE TABLE students(
        regno INT,
        name CHAR(20),
        marks DOUBLE
    );
```

```
CREATE TABLE students_Group(
        regno INT,
        groupname CHAR(10)
    );

    DESCRIBE students; -- see the table structure

    SHOW TABLES; -- dispaly all the tables available in your selected database

    DROP DATABASE sunbeam_stduents;
```

SQL SCRIPTS

```
-- import the classwork database into your mysql

CREATE DATABASE classwork;
USE classwork;
SELECT DATABASE();
SHOW TABLES;

SOURCE <drag drop path to the .sql file> ;
SHOW TABLES;

SELECT * FROM emp;
SELECT * FROM dept;
```

Add and view Data From table

```
CREATE DATABASE my_db;
USE mydb;

CREATE TABLE students(
        regno INT,
        name CHAR(20),
        marks DOUBLE
    );

SHOW TABLES;

DESC students;

--to add the data into table
INSERT INTO students VALUES(1,'stu1',78);

--to display all data from table
SELECT * FROM studnets;
```

Agenda

```
- Data Types
- Char vs Varchar vs TEXT
- DQL
- Computed Columns
- Distinct
- LIMIT
- Order By
- WHERE clause
- Relational Operators
- IN,BETWEEN Operator
```

Steps to change mysql cmd Prompt

```
- Open environment variable
- Inside system variable for rohan click on new
- Enter Variable name as MYSQL_PS1
- Enter value as W1_ROHAN_12345> (Group_name_rollno)
- Click on OK.
```

Datatypes

```
1. Numeric Type
    - tinyint(1 byte)
    - smallint(2 bytes)
    - mediumint(3 bytes)
    - int(4 bytes)
    - bigint(8 bytes)
    - float(4 bytes)
    - double(8 bytes)
    - Decimal(m,n)
        m-> total no of digits
        n-> no of digits after the decimal point
        e.g DECIMAL(4,2) = 12.30
        e.g DECIMAL(5,4) = 1.2345
2. String Type
    - CHAR(n) -> n is the no of characters you want
    - VARCHAR() ->n is the no of characters you want
    - TINYTEXT(255)
    - TEXT(65K)
    - MEDIUMTEXT(16MB)
    - LONGTEXT(4GB)
3. Binary Type
```

```
    - TINYBLOB,BLOB,MEDIUMBLOB,LONGBLOB
4. Misc Type
    - ENUM (M,F,O)-> Radiobutton
    - SET (C,CPP,JAVA) ->Checkbox
5. DateTime Type
    - DATE -> yyyy-mm-dd (1000-01-01 to 9999-12-31)
    - TIME -> hr:min:sec (838:59:59)
    - DATETIME -> yyyy-mm-dd hr:min:sec (1000-01-01 to 9999-12-31)
                                        (00:00:00 to 23:59:59)
    - YEAR - 1901 to 2155
```

Char vs Varchar vs TEXT

```sql
CREATE TABLE temp(
   c1 CHAR(4),
   c2 VARCHAR(4),
   c3 TEXT(4)
);

 DESC temp;

INSERT INTO temp VALUES('ab','ab','ab');
SELECT * FROM temp;

INSERT INTO temp VALUES('abc','abc','abc');
SELECT * FROM temp;

INSERT INTO temp VALUES('abcd','abcd','abcd');
SELECT * FROM temp;

INSERT INTO temp VALUES('abcde','abcd','abcd');--error

INSERT INTO temp VALUES('abcd','abcde','abcd');--error

INSERT INTO temp VALUES('abcd','abcd','abcde');

INSERT INTO temp VALUES('abcde','abcde','abcde');
```

DQL -> SELECT

```sql
USE classwork;
SELECT DATABASE();
SHOW TABLES;

SELECT * FROM emp;
SELECT * FROM dept;
```

```
--dispaly emp name,job,sal from emp table
SELECT ename,job,sal FROM emp;
SELECT ename,sal,job FROM emp;
```

```
--add new emp with name as Rohan sal as 3000 and job as analyst
INSERT INTO emp VALUES('Rohan',3000,'ANALYST');--error
INSERT INTO emp(ename,sal,job) VALUES('Rohan',3000,'ANALYST');

--add 2 new emps in single query
INSERT INTO emp(ename,sal,job)VALUES
('Pratik',2000,'ANALYST'),
('Onkar',2500,'CLERK');
```

DQL- Computed Column

```
-- give DA as an allowance for all employees. It should be 50% of sal
-- dispaly all emps along witn their allowance

--Computed Column
SELECT ename,sal,sal*.50 FROM emp;

--Column Alias
SELECT ename,sal,sal*.50 AS DA FROM emp;

--display gross_sal = sal+DA from emp.
SELECT ename,sal,sal*0.5 AS DA, sal+sal*0.5 AS gross_sal FROM emp;

SELECT ename,sal,sal*0.5 AS DA, sal+DA AS gross_sal FROM emp; --error

--dispaly emp and their deptnames
SELECT ename,deptno, CASE
WHEN deptno=10 THEN 'ACCOUNTING'
WHEN deptno=20 THEN 'RESEARCH'
WHEN deptno=30 THEN 'SALES'
END
AS dept_name
FROM emp;
```

DQL -> Distinct

```
--display all unique jobs from emp.
SELECT job FROM emp;
--It will fetch all jobs from emp table along with repetation

SELECT DISTINCT job FROM emp;
```

```
--display all unique deptno from emp.
SELECT deptno FROM emp;

SELECT DISTINCT deptno FROM emp;

--display unique jobs from each dept
SELECT DISTINCT deptno,job FROM emp;
```

DQL-> Limit

```
--display only 5 employees
SELECT * FROM emp;
SELECT * FROM emp LIMIT 5;

--display only 10 employees
SELECT * FROM emp LIMIT 10;

--display only 5 rows with emp name,sal and job from emp
SELECT ename,sal,job FROM emp;
SELECT ename,sal,job FROM emp LIMIT 5;

--diaply empname,sal,comm from emp skip 5 rows and show 3 rows after that
SELECT ename,sal,comm FROM emp;
SELECT ename,sal,comm FROM emp LIMIT 5;
SELECT ename,sal,comm FROM emp LIMIT 5,3;
```

Order BY

```
-- dispaly emp with sal sorted in ascending manner
SELECT * FROM emp;
SELECT * FROM emp ORDER BY sal; --Default sorting is Ascending

-- dispaly emp with deptno sorted in ascending manner
SELECT * FROM emp;
SELECT * FROM emp ORDER BY deptno;

-- dispaly emp with sal sorted in descending manner
SELECT * FROM emp ORDER BY sal DESC;

-- dispaly emps sorted on their deptno and their jobs
SELECT ename,deptno,job FROM emp;
SELECT ename,deptno,job FROM emp ORDER BY deptno,job;

-- dispaly emps sorted on their deptno asc and their jobs in desc
SELECT ename,deptno,job FROM emp ORDER BY deptno ,job DESC;
```

ORDER By using Limit

```
--display top 3 emps as per highest sal
SELECT * FROM emp;
SELECT * FROM emp ORDER BY sal DESC;
SELECT * FROM emp ORDER BY sal DESC LIMIT 3;

--display single emp which comes last in alphabetical order.
SELECT * FROM emp;
SELECT * FROM emp ORDER BY ename;
SELECT * FROM emp ORDER BY ename DESC;
SELECT * FROM emp ORDER BY ename DESC LIMIT 1;

--display single emp with lowest salary
SELECT * FROM emp ORDER BY sal LIMIT 1;

--display single emp with 3rd lowest salary
SELECT * FROM emp ORDER BY sal;
SELECT * FROM emp ORDER BY sal LIMIT 2,1;

--dispaly single emp with 2nd highest salaray
SELECT * FROM emp ORDER BY sal DESC;
SELECT * FROM emp ORDER BY sal DESC LIMIT 1,1;

--dispaly empname,DA from emop sorted based on DA
SELECT ename,sal*0.5 AS DA FROM emp;
SELECT ename,sal*0.5 FROM emp ORDER BY sal*0.5;
SELECT ename,sal*0.5 AS DA FROM emp ORDER BY DA;
SELECT ename,sal*0.5 FROM emp ORDER BY 2;
```

WHERE CLAUSE

```
-- dispaly emps from dept no 30
SELECT * FROM emp;
SELECT * FROM emp WHERE deptno=30;

--display all emps with sal<2000
SELECT * FROM emp;
SELECT * FROM emp WHERE sal<2000;

--display all emp working as ANALYST
SELECT * FROM emp;
SELECT * FROM emp WHERE job="ANALYST";

--display all emps not working in dept 30
SELECT * FROM emp;
SELECT * FROM emp WHERE deptno!=30;
```

```
SELECT * FROM emp WHERE deptno<>30;
SELECT * FROM emp WHERE NOT deptno=30;

--dispaly all emp who are not salesman
SELECT * FROM emp WHERE job = 'SALESMAN';
SELECT * FROM emp WHERE job != 'SALESMAN';
SELECT * FROM emp WHERE job <> 'SALESMAN';
SELECT * FROM emp WHERE NOT job = 'SALESMAN';

-- display all emp who work as manager and analyst
SELECT * FROM emp;
SELECT * FROM emp WHERE job = 'MANAGER';
SELECT * FROM emp WHERE job = 'ANALYST';
SELECT * FROM emp WHERE job = 'MANAGER' OR job = 'ANALYST';
SELECT * FROM emp WHERE job IN ('MANAGER','ANALYST');

--display all emp in sal range 1000 to 2000
SELECT * FROM emp;
SELECT * FROM emp WHERE sal>=1000;
SELECT * FROM emp WHERE sal<=2000;
SELECT * FROM emp WHERE sal>=1000 AND sal<=2000;
SELECT * FROM emp WHERE sal BETWEEN 1000 AND 2000;

--display emps hired in 1981
SELECT * FROM emp;
SELECT * FROM emp WHERE hire='1981';--error
--1981-01-01 -> 1981-12-31
SELECT * FROM emp WHERE hire>='1981-01-01';
SELECT * FROM emp WHERE hire<='1981-12-31';
SELECT * FROM emp WHERE hire>='1981-01-01' AND hire<='1981-12-31';
SELECT * FROM emp WHERE hire BETWEEN '1981-01-01' AND '1981-12-31';
```

Agenda

```
- Where clause
    - LIMIT
    - LIKE
- DML - Update,Delete
- DDL - Drop,Truncate
- DUAL
- SQL FUNCTIONS
    - String
    - Numeric
    - Date and Time Functions
```

Where Clause for NULL Condition

```
--display all emps with comm as null
SELECT * FROM emp;
SELECT * FROM emp WHERE comm=NULL; -- empty set
SELECT * FROM emp WHERE comm IS NULL;
SELECT * FROM emp WHERE comm <=> NULL;

--display all emps with comm as not null
SELECT * FROM emp WHERE comm!=NULL; -- empty set
SELECT * FROM emp WHERE comm IS NOT NULL;
SELECT * FROM emp WHERE NOT (comm IS NULL);
```

Extra Examples

```
--Insert into emp 3 new emps with name as 'B','J' and 'K'
INSERT INTO emp(ename) VALUES('B'),('J'),('K');

--dispaly all emps whose first letter of name is in the range of B to J
SELECT * FROM emp;
SELECT * FROM emp WHERE ename>='B';
SELECT * FROM emp WHERE ename<='J';
SELECT * FROM emp WHERE ename BETWEEN 'B' AND 'J';
SELECT * FROM emp WHERE ename BETWEEN 'B' AND 'K' AND ename!='K';

--display all emps with sal who are not in range of 1000 to 2000
SELECT * FROM emp;
SELECT * FROM emp WHERE sal>=1000;
SELECT * FROM emp WHERE sal<=2000;
SELECT * FROM emp WHERE NOT sal BETWEEN 1000 AND 2000;

--display all emps between B to J and T to Z
SELECT * FROM emp;
```

```
SELECT * FROM emp WHERE ename BETWEEN 'B' AND 'K' AND ename!='K';
SELECT * FROM emp WHERE ename>='T';

SELECT * FROM emp WHERE ename BETWEEN 'B' AND 'K' AND ename!='K' OR ename>='T';
```

LIKE

```
- We have 2 wildcard characters which we can use with LIKE Operator
- % -> Any no of characters or even Empty
- _ -> Single Occurance of character
```

```
--dispaly all emps with name starting with M
SELECT * FROM emp;
SELECT * FROM emp WHERE ename>='M' AND ename<'N';
SELECT * FROM emp WHERE ename LIKE 'M';
SELECT * FROM emp WHERE ename LIKE 'M%';

--dispaly all emps with name starting with H
SELECT * FROM emp WHERE ename LIKE 'H%';

--dispaly all emps with name ending with H
SELECT * FROM emp WHERE ename LIKE '%H';

--diplay all emps having 'U' in their name
SELECT * FROM emp WHERE ename LIKE '%U%';

--display all emps having 'A' twice
SELECT * FROM emp WHERE ename LIKE '%A%A%';

--display all emps between B to J
SELECT * FROM emp WHERE ename BETWEEN 'B' AND 'J';
SELECT * FROM emp WHERE ename LIKE 'J%';
SELECT * FROM emp WHERE ename BETWEEN 'B' AND 'J' OR ename LIKE 'J%';

--display all emps with 4 letter name
SELECT * FROM emp WHERE ename LIKE '____';

--dispaly all emps having 'R' as the 3rd letter in their name
SELECT * FROM emp WHERE ename LIKE '__R%';

--display emps with 4 letter word with 'R' in the 3rd Position
SELECT * FROM emp WHERE ename LIKE '__R_';
```

Practice Examples

```
--display emp highest salary in range of 1000 to 2000
SELECT * FROM emp;
SELECT * FROM emp WHERE sal BETWEEN 1000 AND 2000;
SELECT * FROM emp WHERE sal BETWEEN 1000 AND 2000 ORDER BY sal DESC;
SELECT * FROM emp WHERE sal BETWEEN 1000 AND 2000 ORDER BY sal DESC LIMIT 1;

--display clerk with min sal
SELECT * FROM emp WHERE job='CLERK';
SELECT * FROM emp WHERE job='CLERK' ORDER BY sal;
SELECT * FROM emp WHERE job='CLERK' ORDER BY sal LIMIT 1;

--display fifth lowest salary from dept 20 and 30.
SELECT * FROM emp WHERE deptno IN(20,30);
SELECT * FROM emp WHERE deptno IN(20,30) ORDER BY sal;
SELECT * FROM emp WHERE deptno IN(20,30) ORDER BY sal LIMIT 4,1;
SELECT * FROM emp WHERE deptno IN(20,30) ORDER BY sal LIMIT 4,1;
SELECT DISTINCT sal FROM emp WHERE deptno IN(20,30) ORDER BY sal LIMIT 4,1;
```

DML

```
UPDATE emp SET empno=1000 WHERE ename='B';
UPDATE emp SET empno=1001 WHERE ename='J';
UPDATE emp SET empno=1003 WHERE ename='K';

--give sal hike of 200 to all emps who work as clerk
UPDATE emp SET sal=sal+200 WHERE job='CLERK';

--delete emps with empno as 1001,1000,1003
DELETE FROM emp WHERE empno=1001;
DELETE FROM emp WHERE empno=1000;
DELETE FROM emp WHERE empno=1003;

--delete all clerks
DELETE FROM emp WHERE job='CLERK';
```

DDL

```
--If you want to delete all the data from a table then use Truncate
TRUNCATE emp;

-- If you want to remove entire table along with its structure the use drop
DROP TABLE emp;
```

- DELETE
  - It will delete the rows which matches with the condition
  - You can even delete all data from table without giving condition

    - This can be rolledback

- TRUNCATE
    - It will delete the entire data from the table
    - the table structure will remain as it is.
    - This cannot be rolledback

- DROP
    - It will remove the entire table from the databases.
    - This cannot be rolledback

DUAL

```
- It is a single row,single column virtual table
```

```
SELECT * FROM books;
SELECT DATABASE();
SELECT DATABASE() FROM DUAL;

SELECT 345*345;
SELECT 345*345 FROM DUAL;

SELECT "HELLO WORLD" AS hello;
SELECT "HELLO WORLD"AS hello FROM DUAL;
```

SQL Functions

```
- HELP FUNCTIONS
```

1. String Funcntion

```
HELP String Functions;

SELECT UPPER('sunbeam'); --SUNBEAM
SELECT LOWER('SUNBEAM'); --sunbeam

SELECT LOWER(ename) FROM emp;

SELECT LEFT('sunbeam',2);-- su
SELECT RIGHT('sunbeam',3); -- eam

SELECT SUBSTRING('SunBeam',3);
SELECT SUBSTRING('SunBeam',2,3)
```

```
SELECT SUBSTRING('SunBeam',-5);
SELECT SUBSTRING('SunBeam',-5,2);

--display emp-job like this -> SMITH-ANALYST
SELECT CONCAT(ename,'-',job) FROM emp;
SELECT CONCAT(ename,'-',job) AS emp_job FROM emp;

--display o/P as -> smith is working as ANALYST
SELECT CONCAT(ename,' is working as ',job) as emp_job FROM emp;
SELECT CONCAT(LOWER(ename),' is working as ',job) as emp_job FROM emp;

--display 1st letter of emp as upper and rest all as lower
SELECT UPPER(LEFT(ename,1)) FROM emp;
SELECT LOWER(SUBSTRING(ename,2)) FROM emp;
SELECT CONCAT(UPPER(LEFT(ename,1)),LOWER(SUBSTRING(ename,2))) AS ename FROM emp;

SELECT LENGTH('sunbeam');

SELECT TRIM('    sunbeam  ');

SELECT LENGTH(TRIM('    sunbeam  '));

SELECT LPAD('9388',10,'X');

SELECT RPAD('9388',10,'X');

--Homework -> 1234 XXXX XXXX 4567
```

2. Numeric Functions

```
HELP Numeric Functions;

SELECT POW(5,3);

SELECT SQRT(25);

SELECT ROUND(123.45);
SELECT ROUND(123.55,1);
SELECT ROUND(123.55,2);
SELECT ROUND(123.45,-1);
SELECT ROUND(123.45,-2);
SELECT ROUND(167.45,-1);

SELECT CEIL(58.25);
SELECT FLOOR(58.25);
```

3. DATE and TIME Functions

```
HELP Date and Time Functions

SELECT NOW();
SELECT SYSDATE();

SELECT NOW(),SLEEP(2),NOW();
--NOW() is going to give the time when quesy gets execute

SELECT SYSDATE(),SLEEP(2),SYSDATE();
--SYSDATE() is going to give the time when the function gets called.

SELECT DATE(NOW());
SELECT TIME(NOW());

SELECT DATE_ADD(NOW(),INTERVAL 84 DAY);

SELECT DATE_ADD(NOW(),INTERVAL 1 MONTH);

SELECT DATE_ADD(NOW(),INTERVAL 1 YEAR);

--to display experiance in no of days as o/p
SELECT ename,hire,DATEDIFF(NOW(),hire) FROM emp;

SELECT ename,hire,TIMESTAMPDIFF(MONTH,hire,NOW()) FROM emp;

SELECT ename,hire,TIMESTAMPDIFF(YEAR,hire,NOW()) FROM emp;

--display emp,hire,experiance in terms of total year and months
SELECT ename,hire,TIMESTAMPDIFF(YEAR,hire,NOW()) AS
YEAR,TIMESTAMPDIFF(MONTH,hire,NOW())%12 AS MONTH FROM emp;

SELECT DAY(NOW()),MONTH(NOW()),YEAR(NOW());
SELECT HOUR(NOW()),MINUTE(NOW()),SECOND(NOW());

--display emps hired in 1981
SELECT * FROM emp WHERE YEAR(hire) = 1981;
```

Agenda

```
- Flow Control Functions
- Group Functions
- Group By
- Having Clause
- Joins - Relationships
```

4. Flow Control Functions

```
 HELP FLOW CONTROL FUNCTIONS

--display ename,deptname from emp.
-- 10 | ACCOUNTING, 20 | RESEARCH
SELECT ename,deptno FROM emp;

SELECT ename,deptno,CASE
WHEN deptno=10 THEN 'ACCOUNTING'
WHEN deptno=20 THEN 'RESEARCH'
ELSE 'OTHER'
END AS dname
FROM emp;

--dipaly emp as RICH if sal is more that 2500 and POOR if it is less than it.
SELECT ename,sal FROM emp;
SELECT ename,sal,IF(sal>2500,'RICH','POOR') AS Category FROM emp;

SELECT ename,sal,comm FROM emp;
SELECT ename,sal,comm,sal+comm AS Total_Income FROM emp;
SELECT ename,sal,comm,IFNULL(comm,0),sal+IFNULL(comm,0) AS Total_Income FROM emp;

--dispaly tax as null is sal is 800
SELECT ename,sal FROM emp;
SELECT ename,sal,NULLIF(sal,800) AS tax_income FROM emp;
```

LIST Functions

```
SELECT CONCAT ('A','B','C','D',3.25);
SELECT CONCAT ('A','B',NULL,'D',3.25);
SELECT LEAST(100,200,50,25);
SELECT LEAST(100,NULL,50,25);
SELECT GREATEST(100,200,50,25);
SELECT GREATEST(100,NULL,50,25);
```

Group Functions

```
- Single Row Function
    n input rows -> n output rows

- Group Function
    n input rows -> 1 output row
    Null values are ignored in Group functions
```

```
HELP Aggregate Functions and Modifiers

-- display empcount,total spending on sal, avg sal,min sal,max sal.

SELECT COUNT(empno),SUM(sal),AVG(sal),MAX(sal),MIN(sal) FROM emp;

SELECT COUNT(*),SUM(sal),AVG(sal),MAX(sal),MIN(sal) FROM emp;


SELECT COUNT(comm),SUM(comm),AVG(comm),MAX(comm),MIN(comm) FROM emp;
```

Limitations of Group Functions

```
- SELECT @@sql_mode;
- Observe this -> ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION
- open notepad with administrator privileges
- Open my.ini file in your notepad.
- this file is available under C:\ProgramData\MySQL\MySQL Server 8.0
- Go under SERVER SECTION -> [mysqld] below this line you will find
- sql-mode="ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION"
- Restart your Mysql Server
```

```
-- display empname and avg sal.
SELECT ename,AVG(sal) FROM emp; --error
-- You cannot select any single column whenever you are using group functions

-- dispaly ename in lower case and avg sal
SELECT LOWER(ename),AVG(sal) FROM emp; --error'
--You cannot use single row functions whenever you are using group functions

--display all emps with Max sal -> MAX()
SELECT * FROM emp WHERE sal = MAX(sal);
--You cannot use group function in where clause

SELECT SUM(MAX(sal)) FROM emp;
--You cannot use nested group function
```

GROUP BY CLAUSE

```
--dispaly dept wise total emp working in that dept and total sal spending
SELECT deptno,SUM(sal) AS Total_sal,COUNT(*) AS emp_count FROM emp; -- error

SELECT deptno,SUM(sal) AS Total_sal,COUNT(*) AS emp_count FROM emp GROUP BY
deptno;

SELECT deptno,SUM(sal) AS Total_sal,COUNT(*) AS emp_count FROM emp GROUP BY deptno
ORDER BY deptno;

--dispaly job wise total emp working in that job and total sal spending
SELECT * FROM emp ORDER BY job;
SELECT SUM(sal),COUNT(*) FROM emp GROUP BY job;
SELECT job,SUM(sal),COUNT(*) FROM emp GROUP BY job;

--diplay the deptno and the min sal of that dept
SELECT MIN(sal) FROM emp;
SELECT MIN(sal) FROM emp GROUP BY deptno;
SELECT deptno,MIN(sal) FROM emp GROUP BY deptno;

--diplay the job and the min sal for that job
SELECT MIN(sal) FROM emp;
SELECT MIN(sal) FROM emp GROUP BY job;
SELECT job,MIN(sal) FROM emp GROUP BY job;

--display count of emp working in dept with sepecific job
SELECT COUNT(*) FROM emp;
SELECT COUNT(*) FROM emp GROUP BY deptno;
SELECT COUNT(*) FROM emp GROUP BY job;
SELECT COUNT(*) FROM emp GROUP BY deptno,job;
SELECT deptno,COUNT(*) FROM emp GROUP BY deptno,job;
SELECT deptno,job,COUNT(*) FROM emp GROUP BY deptno,job;

SELECT deptno,job,COUNT(*) FROM emp GROUP BY deptno,job ORDER BY deptno;
```

Having Clause

- Having clause must be used only with group by clause
- If their is condition on aggregate/group values

```
-- display deptwise total salary spend if total sal > 9000
SELECT deptno,SUM(sal) FROM emp;
SELECT deptno,SUM(sal) FROM emp GROUP BY deptno;
SELECT deptno,SUM(sal) FROM emp GROUP BY deptno HAVING SUM(sal)>9000;
```

```sql
--display job wise avg sal where the avg sal is > 2500
SELECT job,AVG(sal) FROM emp GROUP BY job;
SELECT job,AVG(sal) FROM emp GROUP BY job HAVING AVG(sal)>2500;

--display max sal of all jobs of dept 10 and 20
SELECT deptno,job,MAX(sal) FROM emp GROUP BY deptno,job;
SELECT deptno,job,MAX(sal) FROM emp GROUP BY deptno,job HAVING deptno IN (10,20);
SELECT deptno,job,MAX(sal) FROM emp WHERE deptno IN(10,20) GROUP BY deptno,job;

--display max sal of all jobs of dept 10 and 20 where max sal > 2500
SELECT deptno,job,MAX(sal) FROM emp
WHERE deptno IN(10,20)
GROUP BY deptno,job
HAVING MAX(sal)>2500;

--dispaly only one dept that spends maximum on emps salary.
SELECT deptno,SUM(sal) FROM emp GROUP BY deptno;
SELECT deptno,SUM(sal) FROM emp GROUP BY deptno ORDER BY SUM(sal) DESC;
SELECT deptno,SUM(sal) FROM emp GROUP BY deptno ORDER BY SUM(sal) DESC LIMIT 1;

--dispaly job having lowest avg(sal)
SELECT job,AVG(sal) FROM emp GROUP BY job;
SELECT job,AVG(sal) FROM emp GROUP BY job ORDER BY AVG(sal);
SELECT job,AVG(sal) FROM emp GROUP BY job ORDER BY AVG(sal) LIMIT 1;

--display job having lowest avg income
SELECT job,AVG(sal+IFNULL(comm,0)) AS income FROM emp
GROUP BY job
ORDER BY income
LIMIT 1;
```

Agenda

```
-JOINS
```

JOINS

1. CROSS JOIN

```
--display ename,deptname from emps and depts
SELECT e.ename,d.dname FROM emps e CROSS JOIN depts d;

SELECT ename,dname FROM emps CROSS JOIN depts;

SELECT ename,deptno,dname FROM emps CROSS JOIN depts; --error

SELECT ename,e.deptno,dname FROM emps e CROSS JOIN depts d;
```

2. INNER JOIN

```
--display ename,deptname from emps and depts
SELECT e.ename,d.dname FROM emps e INNER JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname FROM depts d INNER JOIN emps e ON e.deptno=d.deptno;
-- equi joins

SELECT e.ename,d.dname FROM emps e INNER JOIN depts d ON e.deptno!=d.deptno;
--non-equi joins
```

3. LEFT OUTER JOIN
    - LEFT OUTER JOIN = INTERESECTION + DATA FROM LEFT TABLE

```
--display ename,deptname from emps and depts
SELECT e.ename,d.dname FROM emps e LEFT OUTER JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname FROM depts d LEFT OUTER JOIN emps e ON e.deptno=d.deptno;
```

4. RIGHT OUTER JOIN
    - RIGHT OUTER JOIN = INTERESECTION + DATA FROM RIGHT TABLE

```
--display ename,deptname from emps and depts
SELECT e.ename,d.dname FROM emps e RIGHT OUTER JOIN depts d ON e.deptno=d.deptno;
```

```
SELECT e.ename,d.dname FROM depts d RIGHT OUTER JOIN emps e ON e.deptno=d.deptno;
```

5. FULL OUTER JOIN
    - FULL OUTER JOIN = LEFT OUTER JOIN + RIGHT OUTER JOIN
    - This is not supported in mysql
    - But however this can be implemented by using set operators

```
SELECT e.ename,d.dname FROM emps e LEFT OUTER JOIN depts d ON e.deptno=d.deptno;
SELECT e.ename,d.dname FROM emps e RIGHT OUTER JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname FROM emps e LEFT OUTER JOIN depts d ON e.deptno=d.deptno
UNION
SELECT e.ename,d.dname FROM emps e RIGHT OUTER JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname FROM emps e LEFT OUTER JOIN depts d ON e.deptno=d.deptno
UNION ALL
SELECT e.ename,d.dname FROM emps e RIGHT OUTER JOIN depts d ON e.deptno=d.deptno;
```

6. SELF JOIN

```
--display ename and manager name of that emp.
SELECT e.ename,m.ename AS mname FROM emps e INNER JOIN emps m ON e.mgr=m.mgr;
--wrong o/p

SELECT e.ename,m.ename AS mname FROM emps e INNER JOIN emps m ON e.mgr=m.empno;

SELECT e.ename,m.ename AS mname FROM emps e LEFT JOIN emps m ON e.mgr=m.empno;
```

JOINS PRACTICE

```
SELECT * FROM emps;
SELECT * FROM depts;
SELECT * FROM meeting;
SELECT * FROM addr;
SELECT * FROM emp_meeting;

--display empno,ename,deptno and deptname of all employees
SELECT e.empno,e.ename,e.deptno,d.dname FROM emps e
INNER JOIN depts d ON e.deptno=d.deptno;

SELECT e.empno,e.ename,e.deptno,d.dname FROM emps e
LEFT JOIN depts d ON e.deptno=d.deptno;
```

```sql
--display ename,deptname and address of that employee
SELECT e.ename,d.dname FROM emps e
INNER JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,a.tal,a.dist FROM emps e
INNER JOIN addr a ON e.empno=a.empno;

SELECT e.ename,a.tal,a.dist,d.dname FROM emps e
INNER JOIN addr a ON e.empno=a.empno
INNER JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname,a.tal,a.dist FROM emps e
INNER JOIN addr a ON e.empno=a.empno
LEFT JOIN depts d ON e.deptno=d.deptno;

--display emp and their meeting topics
SELECT * FROM emp;
SELECT * FROM meeting;
SELECT * FROM emp_meeting;

SELECT e.ename,em.meetno FROM emps e
INNER JOIN emp_meeting em ON e.empno=em.empno;

SELECT m.topic,em.empno FROM meeting m
INNER JOIN emp_meeting em ON m.meetno=em.meetno;

SELECT e.ename,m.topic FROM emps e
INNER JOIN emp_meeting em ON e.empno=em.empno
INNER JOIN meeting m ON m.meetno=em.meetno;

-- display empname,meeting topic and his address
SELECT e.ename,m.topic FROM emps e
INNER JOIN emp_meeting em ON e.empno=em.empno
INNER JOIN meeting m ON m.meetno=em.meetno;

SELECT e.ename,a.tal,a.dist FROM emps e
INNER JOIN addr a ON e.empno=a.empno;

SELECT e.ename,m.topic,a.tal,a.dist FROM emps e
INNER JOIN emp_meeting em ON e.empno=em.empno
INNER JOIN meeting m ON m.meetno=em.meetno
INNER JOIN addr a ON e.empno=a.empno;

-- display empname,deptname,meeting topic and his address
SELECT e.ename,m.topic,a.tal,a.dist FROM emps e
INNER JOIN emp_meeting em ON e.empno=em.empno
INNER JOIN meeting m ON m.meetno=em.meetno
INNER JOIN addr a ON e.empno=a.empno;

SELECT e.ename,d.dname FROM emps e
INNER JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname,m.topic,a.tal,a.dist FROM emps e
INNER JOIN emp_meeting em ON e.empno=em.empno
```

```
INNER JOIN meeting m ON m.meetno=em.meetno
INNER JOIN addr a ON e.empno=a.empno
LEFT JOIN depts d ON e.deptno=d.deptno;

--print deptname and count of employees in that dept
SELECT * FROM emps;
SELECT deptno,COUNT(empno) FROM emps GROUP BY deptno;

SELECT d.dname,COUNT(e.empno) FROM emps e
INNER JOIN depts d ON e.deptno=d.deptno
GROUP BY d.dname;

SELECT d.dname,COUNT(e.empno) FROM emps e
RIGHT JOIN depts d ON e.deptno=d.deptno
GROUP BY d.dname;

--display emps and their total meetings in desc order of their meeting count
SELECT e.ename,em.meetno FROM emps e
INNER JOIN emp_meeting em ON e.empno=em.empno;

SELECT e.ename,COUNT(em.meetno) AS emp_count FROM emps e
INNER JOIN emp_meeting em ON e.empno=em.empno
GROUP BY e.ename
ORDER BY emp_count DESC;

--display all emps from 'dev' dept
SELECT e.ename,d.dname FROM emps e
INNER JOIN depts d ON e.deptno=d.deptno
WHERE d.dname='DEV';
```

Usage of all Clauses as per priority

```
SELECT columns FROM table1
XXX JOIN table2 XXX ON condition
XXX JOIN table3 XXX ON condition
WHERE condition
GROUP BY column
Having condition
ORDER BY column
LIMIT n
```

Non standard joins

```
--display ename,dname of emps.
SELECT e.ename,d.dname FROM emps e
INNER JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname FROM emps e
```

```
JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname FROM emps e
CROSS JOIN depts d ON e.deptno=d.deptno;

SELECT e.ename,d.dname FROM emps e
CROSS JOIN depts d WHERE e.deptno=d.deptno;

SELECT e.ename,d.dname FROM emps e,
depts d WHERE e.deptno=d.deptno;

SELECT e.ename,d.dname FROM emps e
INNER JOIN depts d USING(deptno);
--this join only works in mysql

SELECT e.ename,d.dname FROM emps e
NATURAL JOIN depts d;
```

Agenda

```
Security
Transactions
Row Locking
```

Security

```
mysql> PROMPT \u>

root>SELECT user FROM mysql.user;

root>CREATE USER mgr IDENTIFIED BY 'mgr';

root>SELECT user FROM mysql.user;

root>SHOW DATABASES;

--to give permissions to mgr on specific table
root>GRANT ALL PRIVILEGES ON classwork.emp TO mgr;

--open new cmd prompt
cmd>mysql -u mgr -pmgr

mgr>SHOW DATABASES;

mgr>SHOW TABLES;

root>CREATE USER teamlead@localhost IDENTIFIED BY 'team';

root>SELECT user,host FROM mysql.user;

--open new cmd prompt
cmd>mysql -u teamlead -pteam

teamlead>SHOW DATABASES;

mgr>SHOW GRANTS;

root>SHOW GRANTS FOR mgr;

root>SHOW GRANTS FOR teamlead@localhost;

mgr>GRANT ALL PRIVILEGES ON classwork.emp TO teamlead@localhost; -- error

root>GRANT ALL PRIVILEGES ON classwork.* TO mgr WITH GRANT OPTION;

mgr>GRANT ALL PRIVILEGES ON classwork.emp TO teamlead@localhost; --OK
```

```
mgr>GRANT ALL PRIVILEGES ON classwork.dept TO teamlead@localhost;

root>CREATE USER dev1 IDENTIFIED BY 'dev1';

--open new cmd prompt
cmd>mysql -u dev1 -pdev1

mgr>GRANT SELECT ON classwork.emp TO dev1;

mgr>GRANT INSERT,UPDATE ON classwork.emp TO dev1;

mgr>REVOKE UPDATE ON classwork.emp FROM dev1;

root>DROP USER dev1;
```

Transactions - TCL

```
TCL COMMANDS
    - START TRANSACTION
    - ROLLBACK
    - COMMIT
```

```
CREATE TABLE accounts(
    id INT,
    type CHAR(10),
    balance DECIMAL(9,2)
);

INSERT INTO accounts VALUES
(1,'SAVINGS',2000),
(2,'CURRENT',3000),
(3,'SAVINGS',4000),
(4,'CURRENT',5000),
(5,'SAVINGS',6000);

SELECT * FROM accounts;

root>UPDATE accounts set balance=balance-1000 WHERE id=5;
root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;

root>START TRANSACTION
root>UPDATE accounts set balance=balance-1000 WHERE id=4;
root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;
root>ROLLBACK;
root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;
```

```
root>START TRANSACTION
root>UPDATE accounts set balance=balance-1000 WHERE id=4;
root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;
root>COMMIT;
root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;
```

SAVEPOINT

```
- Save point can be called as a state in a transaction
```

```
root>START TRANSACTION;
root>INSERT INTO accounts VALUES(6,"TEMP",1000);
root>SELECT * FROM acounts;
root>SAVEPOINT sp1;

root>DELETE FROM accounts WHERE id=4;
root>DELETE FROM accounts WHERE id=5;
root>SELECT * FROM acounts;
root>SAVEPOINT sp2;

root>UPDATE accounts SET balance=balance+2000 WHERE id=1;
root>SELECT * FROM acounts;

root>ROLLBACK TO sp2;
root>SELECT * FROM accounts;

root>ROLLBACK;
root>SELECT * FROM accounts;
```

Transaction Properties

```
A - Atomicity
    DML opertaions will either be successfull/Failure. Partial transactions are
never commited.
C - Consistency
    At the end of transaction same state will be visible to all the users.
I - Isolation
    Every transaction is isolated from each other.
D - Durability
    At the end of transaction the state will be saved on to the server.
```

ROW LOCKING

```
root>START TRANSACTION
root>UPDATE accounts set balance=balance-5000 WHERE id=5;
root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;
mgr>UPDATE accounts set balance=balance-5000 WHERE id=5; --terminal hangs
root>ROLLBACK;
root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;
```

TABLE LOCKING

```
If your table does not have a primary key then your whole table gets locked.
```

```
root>START TRANSACTION
root>UPDATE accounts set balance=balance-4000 WHERE id=4;
root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;
mgr>UPDATE accounts set balance=balance-4000 WHERE id=4; --terminal hangs
mgr>UPDATE accounts set balance=balance-3000 WHERE id=2; --terminal hangs

root>SELECT * FROM accounts;
mgr>SELECT * FROM accounts;
```

Pessimistic Locking

```
root>START TRANSACTION;
root>SELECT id,name,subject FROM books WHERE id=2002 FOR UPDATE;

mgr>UPDATE books set subject='D++ Programming' WHERE id=2002;--terminal hangs
```

Agenda

```
SubQuery
```

SubQuery

```
- It is query inside another query
1. Single Row Subquery
2. Multi Row Subquery
```

1. Single Row Subquery

```
--display all emps with maximum sal.
SELECT max(sal) FROM emp;
SELECT * FROM emp ORDER BY sal DESC LIMIT 1;

SELECT * FROM emp WHERE sal=MAX(sal); -- error
SELECT max(sal) FROM emp;
SELECT * FROM emp WHERE sal=5000;

SELECT * FROM emp WHERE sal=(SELECT max(sal) FROM emp);


--display all emps with second highest salary
SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT 1,1;
SELECT * FROM emp WHERE sal=(SELECT DISTINCT sal FROM emp ORDER BY sal DESC LIMIT
1,1);

--display all emps working in dept same as employee KING
SELECT deptno FROM emp WHERE ename='KING';
SELECT * FROM emp WHERE deptno=(SELECT deptno FROM emp WHERE ename='KING');
SELECT * FROM emp WHERE deptno=(SELECT deptno FROM emp WHERE ename='KING') AND
ename<>'KING';

--display all emps working in job same as that of SCOTT
SELECT job FROM emp WHERE ename='SCOTT';
SELECT * FROM emp WHERE job = (SELECT job FROM emp WHERE ename='SCOTT');
SELECT * FROM emp WHERE job = (SELECT job FROM emp WHERE ename='SCOTT') AND
ename!='SCOTT';
```

2. Multi Row Subquery
    - The inner query return multiple rows.
    - the multiple rows from inner query can be compared by using IN,ANY,ALL operators

```
-- display all emps having sal more than all salesman
SELECT sal FROM emp where job='SALESMAN';
SELECT * FROM emp WHERE sal > ALL(SELECT sal FROM emp where job='SALESMAN');
--(sal>1600 AND sal>1250 AND sal>1500)

-- display all emps having sal less than any dept of 20
SELECT sal FROM emp WHERE deptno=20;
SELECT * FROM emp WHERE sal< ANY(SELECT sal FROM emp WHERE deptno=20);
--(sal<800 OR sal<2975 OR sal< 3000 OR...)

--display all dept name which have employees
SELECT DISTINCT deptno FROM emp;
SELECT * FROM dept WHERE deptno= ANY(SELECT DISTINCT deptno FROM emp);

SELECT * FROM dept WHERE deptno IN(SELECT DISTINCT deptno FROM emp);

--display all dept name which dont have employees

SELECT * FROM dept WHERE deptno != ALL(SELECT DISTINCT deptno FROM emp);
SELECT * FROM dept WHERE deptno NOT IN(SELECT DISTINCT deptno FROM emp);
```

ALL, ANY vs IN Operator

```
-ANY can be used only with subquery, In can be used with/without subquery
-ANY can be use with all relational operators,IN can be used with only for equal
condition
-ANY behaves like logical OR
-ALL can be used only with sub query
-ALL can be used with all relational operators
-All behaves like logical AND
```

Corelated Subquery

```
If your inner query is having a condition(Where clause) based on current row of
outer query then such type of query is called as corelated subquery
```

```
--display all dept name which have employees
SELECT * FROM dept WHERE deptno= ANY(SELECT deptno FROM emp);
--10 - ACC - SELECT deptno FROM emp;- 13 rows
--20 - RES - SELECT deptno FROM emp;- 13 rows
--30 - SAL - SELECT deptno FROM emp;- 13 rows
--40 - OPS - SELECT deptno FROM emp;- 13 rows

--display all dept name which have employees
SELECT * FROM dept WHERE deptno= ANY(SELECT DISTINCT deptno FROM emp);
```

```
--10 - ACC - SELECT DISTINCT deptno FROM emp;-10,20,30
--20 - RES - SELECT DISTINCT deptno FROM emp;-10,20,30
--30 - SAL - SELECT DISTINCT deptno FROM emp;-10,20,30
--40 - OPS - SELECT DISTINCT deptno FROM emp;-10,20,30

SELECT * FROM dept d WHERE d.deptno=ANY
(SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno);
--10 - ACC - SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno-10,10,10
--20 - RES - SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno-20 ->5 rows
--30 - SAL - SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno-30 -> 6 rows
--40 - OPS - SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno-0 rows

SELECT * FROM dept d WHERE d.deptno=
(SELECT DISTINCT e.deptno FROM emp e WHERE e.deptno=d.deptno);
--10 - ACC - SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno-10
--20 - RES - SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno-20
--30 - SAL - SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno-30
--40 - OPS - SELECT e.deptno FROM emp e WHERE e.deptno=d.deptno-0 rows
```

Subquery in Projection

```
--display dept wise count of employees and total no of employees
--deptno - countof emp - total emp
--20          5                14
--30          6                14
--10          3                14

SELECT deptno,COUNT(empno) FROM emp;--error
SELECT deptno,COUNT(empno) FROM emp GROUP BY deptno;
SELECT COUNT(empno) FROM emp;

SELECT deptno,COUNT(empno),(SELECT COUNT(empno) FROM emp) FROM emp GROUP BY
deptno;

SELECT deptno,COUNT(empno) AS empcount,(SELECT COUNT(empno) FROM emp) AS
totalcount FROM emp GROUP BY deptno;
```

Subquery in FROM clause

```
- If subquery is written in FROM clause then it should compulsary have an alias
- The subquery in from clause is called as derived table or inline view.
```

```
--diplay emp as rich is sal>2000 and poor if sal<2000
SELECT ename,sal,CASE
WHEN sal>=2000 THEN 'RICH'
WHEN sal<2000 THEN 'POOR'
```

```
END
AS category
FROM emp;

-- display count of employess in above category
SELECT category,COUNT(ename) FROM
(SELECT ename,sal,CASE
WHEN sal>=2000 THEN 'RICH'
WHEN sal<2000 THEN 'POOR'
END
AS category
FROM emp) AS category_count
GROUP BY category;
```

Subquery in DML operations

```
--insert employee with dept as operations
SELECT deptno FROM dept WHERE dname='OPERATIONS';

INSERT INTO emp(empno,ename,sal,deptno) VALUES
(1001,'Rohan',1234,(SELECT deptno FROM dept WHERE dname='OPERATIONS'));


--update employee and change sal to 5678 with dept as operations
UPDATE emp set sal=5678 WHERE deptno=(SELECT deptno FROM dept WHERE
dname='OPERATIONS');

--delete all emps from operations department
DELETE FROM emp WHERE deptno=(SELECT deptno FROM dept WHERE dname='OPERATIONS');

--delete from emp having max sal.
DELETE FROM emp WHERE sal=(SELECT MAX(sal) FROM emp);
--error bcoz you cant select from same table on which you are performing DML
```

SQL PERFORMANCE

```
--display all emps having sal < max of sal
SELECT * FROM emp WHERE sal < (SELECT max(sal) FROM emp);

--display all emps having sal < max of sal from dept 20
SELECT * FROM emp WHERE sal < (SELECT max(sal) FROM emp WHERE deptno=20);

SELECT * FROM emp WHERE sal < ANY(SELECT sal FROM emp WHERE deptno=20);

EXPLAIN FORMAT=JSON SELECT * FROM emp WHERE sal < ANY(SELECT sal FROM emp WHERE
deptno=20);

EXPLAIN FORMAT=JSON SELECT * FROM emp WHERE sal < (SELECT max(sal) FROM emp WHERE
deptno=20);
```

Agenda

```
- Subquery
    EXISTS
    NOT EXISTS
- VIEWS
- INDEXES
```

```
--display dname in which employees exist
SELECT * FROM dept WHERE deptno =ANY (SELECT deptno FROM emp);
SELECT * FROM dept WHERE deptno =ANY (SELECT DISTINCT deptno FROM emp);
SELECT * FROM dept d WHERE deptno = (SELECT DISTINCT e.deptno FROM emp e WHERE
e.deptno=d.deptno);

SELECT * FROM dept d WHERE EXISTS (SELECT DISTINCT e.deptno FROM emp e WHERE
e.deptno=d.deptno);

--display dname in which employees does't exist
SELECT * FROM dept d WHERE NOT EXISTS (SELECT DISTINCT e.deptno FROM emp e WHERE
e.deptno=d.deptno);
```

VIEWS

```
1. SIMPLE VIEW
    -DQL + DML
2. COMPLEX VIEW
    -DQL

View is a projection of data
```

```
--display emp,sal and their category
SELECT ename,sal,CASE
WHEN sal>2000 THEN 'RICH'
WHEN sal<=2000 THEN 'POOR'
END AS category
FROM emp;

-- create a view for above query
CREATE VIEW v_emp_category AS
SELECT ename,sal,CASE
WHEN sal>2000 THEN 'RICH'
WHEN sal<=2000 THEN 'POOR'
END AS category
FROM emp;
```

```
SHOW TABLES;

SHOW FULL TABLES;

DESC v_emp_category;

SELECT * FROM v_emp_category;

SELECT ename,category FROM v_emp_category;

SELECT category,count(ename) FROM v_emp_category GROUP BY category;

--add one more category as MIDDLE where 1000<sal<2000
SELECT ename,sal,CASE
WHEN sal>2000 THEN 'RICH'
WHEN sal>1000 AND sal<2000 THEN 'MIDDLE'
WHEN sal<=1000 THEN 'POOR'
END AS category
FROM emp;

--alter the v_emp_category view for above requirement
ALTER VIEW v_emp_category AS
SELECT ename,sal,CASE
WHEN sal>2000 THEN 'RICH'
WHEN sal>1000 AND sal<2000 THEN 'MIDDLE'
WHEN sal<=1000 THEN 'POOR'
END AS category
FROM emp;

SHOW TABLES;

SELECT * FROM v_emp_category;

SHOW CREATE VIEW v_emp_category;
--It will show how the view is created

--delete a view
DROP VIEW v_emp_category;
```

```
--create a view on empno,ename,sal
CREATE VIEW v_emp_sal AS
SELECT empno,ename,sal FROM emp;

--create a view on empno,ename,sal where sal>2500
CREATE VIEW v_richemp AS
SELECT empno,ename,sal FROM emp WHERE sal>2500;

--create a view on empno,sal,comm and total income
CREATE VIEW v_totalincome AS
SELECT empno,ename,sal,comm,sal+IFNULL(comm,0) AS total_income FROM emp;
```

```
--create a view on job,count(emps),sum(sal),max,min,avg of sal
CREATE VIEW v_jobsummary AS
SELECT job,COUNT(empno) empcount, SUM(sal) salsum,MAX(sal) maxsal,MIN(sal) minsal,
AVG(sal) avgsal FROM emp GROUP BY job;

INSERT INTO emp(empno,ename,sal,deptno) VALUES(1000,'JILL',2000,40);
```

```
SELECT * FROM v_richemp;

INSERT INTO v_richemp(empno,ename,sal) VALUES(1001,'JAMES',3500);
SELECT * FROM v_richemp; -- james will be visible
SELECT * FROM emp;

INSERT INTO v_richemp(empno,ename,sal) VALUES(1002,'ROCK',2200);
SELECT * FROM v_richemp; -- ROCK will not be visible
SELECT * FROM emp;

ALTER VIEW v_richemp AS
SELECT empno,ename,sal FROM emp WHERE sal>2500
WITH CHECK OPTION;

INSERT INTO v_richemp(empno,ename,sal) VALUES(1003,'HARRY',2100);
-- cannot insert as check option will fail

--can we insert the data into v_jobsummary?
SELECT * FROM v_jobsummary;
```

Creating view from another view

```
CREATE VIEW v_richemp2 AS
SELECT empno,ename,sal FROM v_richemp;

SHOW TABLES;
SHOW CREATE VIEW v_richemp;
SHOW CREATE VIEW v_richemp2;

INSERT INTO v_richemp2(empno,ename,sal) VALUES(1004,'BRUCE',2300);
--error

DROP VIEW v_richemp;

SELECT * FROM v_richemp2;--error
--reference table/view invalid

DROP VIEW v_richemp2;
```

Creating view using join

```
-- display ename,sal,deptno,dname form emp;
SELECT e.ename,e.sal,d.deptno,d.dname FROM emp e
INNER JOIN dept d ON e.deptno=d.deptno;

--create a view for above requirement
CREATE VIEW v_empdept AS
SELECT e.ename,e.sal,d.deptno,d.dname FROM emp e
INNER JOIN dept d ON e.deptno=d.deptno;

--diplay all emps from 'accounting' dept
SELECT e.ename,e.sal,d.deptno,d.dname FROM emp e
INNER JOIN dept d ON e.deptno=d.deptno WHERE d.dname='ACCOUNTING';

SELECT * FROM v_empdept WHERE dname="ACCOUNTING";
```

INDEXES

1. SIMPLE INDEX

```
SELECT * FROM books;

--i want to display all books of C Programming
SELECT * FROM books WHERE subject='C Programming';

EXPLAIN FORMAT=JSON SELECT * FROM books WHERE subject='C Programming';
--1.45

DESC books;

CREATE INDEX idx_books_subject ON books(subject);

DESC books;

SELECT * FROM books WHERE subject='C Programming';
EXPLAIN FORMAT=JSON SELECT * FROM books WHERE subject='C Programming';
--0.90

SELECT * FROM books WHERE author = 'Yashwant Kanetkar';

EXPLAIN FORMAT=JSON SELECT * FROM books WHERE author = 'Yashwant Kanetkar';

CREATE INDEX idx_books_author ON books(author DESC);
DESC books;

--display ename,sal,deptno,dname.
EXPLAIN FORMAT = JSON SELECT e.ename,e.sal,d.deptno,d.dname FROM emp e
INNER JOIN dept d ON e.deptno=d.deptno;
--7.70

CREATE INDEX idx_emp_deptno ON emp(deptno);
```

```
CREATE INDEX idx_dept_deptno ON dept(deptno);

EXPLAIN FORMAT = JSON SELECT e.ename,e.sal,d.deptno,d.dname FROM emp e
INNER JOIN dept d ON e.deptno=d.deptno;
--4.01
```

2. UNIQUE INDEX
    - Duplicate values are not allowed in that column
    - Multiple NULL values are allowed

```
DESC emp;
CREATE UNIQUE INDEX idx_emp_ename ON emp(ename);
DESC emp;

INSERT INTO emp(empno,ename) VALUES(1000,'JAMES');
--error

INSERT INTO emp(empno,sal) VALUES(1000,1000);
INSERT INTO emp(empno,sal) VALUES(1001,2000);

CREATE UNIQUE INDEX idx_emp_mgr ON emp(mgr);
--error cannot create unique index on columns which have duplicate values.

SHOW INDEXES FROM emp;
```

3. CLUSTERED INDEX
    - It is a index that is automatically created on primary key
    - It is a unique index
    - If primary key does not exists in table then clusterd index is created on hidden(synthetic) coloumn

```
SHOW INDEXES FROM emp;
DROP INDEX idx_emp_ename ON emp;

SHOW INDEXES FROM books;
DROP INDEX idx_books_author ON books;
```

Agenda

```
-Indexes
    - Composite Index
- Constraints
    - NOT NULL
    - UNIQUE
    - PRIMARY KEY
    - FOREIGN KEY
    - CHECK
- ALTER
```

4. Composite Index
    - It is a index that is applied on two columns

```
--display emp from dept 20;
SELECT * FROM emp Where deptno=20;

--display emp from job CLERK;
SELECT * FROM emp Where job='CLERK';

--display all emps who are working as clerk in dept 20
SELECT * FROM emp Where deptno=20 AND JOB='CLERK';

EXPLAIN FORMAT=JSON SELECT * FROM emp Where deptno=20 AND JOB='CLERK';
--1.45

CREATE INDEX idx_emp_dj ON emp(deptno ASC,job ASC);

SHOW INDEXES FROM emp

EXPLAIN FORMAT=JSON SELECT * FROM emp Where deptno=20 AND JOB='CLERK';
--0.70

--check query cost for finding emp with sal 5000;
EXPLAIN FORMAT=JSON SELECT * FROM emp WHERE sal=5000;
--1.45

--check query cost for finding emp with dept 20;
EXPLAIN FORMAT=JSON SELECT * FROM emp WHERE deptno=20;
--1.00

--check query cost for finding emp with job clerk;
EXPLAIN FORMAT=JSON SELECT * FROM emp WHERE job='clerk';
--1.45

CREATE INDEX idx_emp_job ON emp(job ASC);

EXPLAIN FORMAT=JSON SELECT * FROM emp WHERE job='clerk';
```

```
--0.90

CREATE TABLE students(std INT, roll INT, name CHAR(30), marks DECIMAL(5,2));
INSERT INTO students VALUES (1, 1, 'Soham', 99);
INSERT INTO students VALUES (1, 2, 'Sakshi', 96);
INSERT INTO students VALUES (1, 3, 'Prisha', 98);
INSERT INTO students VALUES (2, 1, 'Madhu', 97);
INSERT INTO students VALUES (2, 2, 'Om', 95);

CREATE UNIQUE INDEX idx_std_roll ON students(std ASC,roll ASC);
INSERT INTO students VALUES (2, 2, 'Rahul', 95);--error duplicate values
INSERT INTO students VALUES (2, 1, 'Rahul', 95);--error duplicate values
INSERT INTO students VALUES (1, 3, 'Rahul', 95);--error duplicate values

INSERT INTO students VALUES (2, 3, 'Rahul', 95);
```

Constraints

```
- Constraints are checked/verified when you do any DML operations.
- It will slow down the DML operations
- It helps to enter the valid and correct data into the tables.
- You can put the constraints on column level or on table level.
- except NOT NULL all the other constraints can be applied on column level as well
as on table level.
```

1. NOT NULL
    - If you want to check for any column that should not have null values while performing DML we use use NOT NULL Constraint

```
CREATE TABLE temp(c1 INT,c2 INT, c3 INT NOT NULL);
INSERT INTO temp VALUES(1,1,1);
INSERT INTO temp(c2,c3) VALUES(2,2);
INSERT INTO temp(c1,c3) VALUES(3,3);
INSERT INTO temp(c1,c2) VALUES(4,4); --error
```

2. UNIQUE
    - If you want to keep unique vaules in column then we should use unique constraint
    - Unique can have multiple NULL values but repetation of values are not allowed.
    - Unique constraint on combination of multiple columns internally creates Composite Unique index.
    - Must be at table level -- UNIQUE(c1,c2).

```
CREATE TABLE temp1(c1 INT UNIQUE,c2 INT, c3 INT);

DESC temp1;
```

```
SHOW INDEXES FROM temp1;

INSERT INTO temp1 VALUES(1,1,1);
INSERT INTO temp1 VALUES(2,1,1);
SELECT * FROM temp1;
INSERT INTO temp1 VALUES(2,2,2);--error
INSERT INTO temp1(c2,c3) VALUES(3,3);
INSERT INTO temp1(c2,c3) VALUES(4,4);

DROP TABLE students;
CREATE TABLE students(std INT, roll INT, name CHAR(30), marks
DECIMAL(5,2),UNIQUE(std,roll));
--using Unique on table level

INSERT INTO students VALUES (1, 1, 'Soham', 99);
INSERT INTO students VALUES (1, 2, 'Sakshi', 96);
INSERT INTO students VALUES (1, 3, 'Prisha', 98);
INSERT INTO students VALUES (2, 1, 'Madhu', 97);
INSERT INTO students VALUES (2, 2, 'Om', 95);

DESC students;
INSERT INTO students VALUES (2, 2, 'Rahul', 95);--error duplicate values
INSERT INTO students VALUES (2, 1, 'Rahul', 95);--error duplicate values
INSERT INTO students VALUES (1, 3, 'Rahul', 95);--error duplicate values

INSERT INTO students(name,marks) VALUES ('Rahul', 95);
INSERT INTO students(std,name,marks) VALUES (1,'Pratik', 85);
INSERT INTO students(std,name,marks) VALUES (1,'Onkar', 75);
INSERT INTO students VALUES (2, 3, 'Vishaka', 95);
SELECT * FROM students;
```

3. Primary Key
    - Unique + NOT NULL
    - In a table you can a single primary key but you can have multiple unique constraints.

```
CREATE TABLE cdac_students(
prnno INT PRIMARY KEY,
name CHAR(30),
email VARCHAR(50),
mobileno CHAR(10),
marks INT
);

CREATE TABLE temp2(c1 INT PRIMARY KEY,c2 INT,c3 INT);
DESC temp2;
SHOW INDEXES FROM temp2;

INSERT INTO temp2 VALUES(1,1,1);
INSERT INTO temp2 VALUES(2,1,1);
INSERT INTO temp2 VALUES(2,3,3); -- error
INSERT INTO temp2 VALUES(NULL,3,3); -- error
```

Composite Primary Key

The primary key can be combination multiple columns. It is called as Composite Primary Key.

```
DROP TABLE students;

CREATE TABLE students(std INT, roll INT, name CHAR(30), marks DECIMAL(5,2),PRIMARY
KEY(std,roll));

DESC students;

SHOW INDEXES FROM students;

INSERT INTO students VALUES (1, 1, 'Soham', 99);
INSERT INTO students VALUES (1, 2, 'Sakshi', 96);
INSERT INTO students VALUES (1, 3, 'Prisha', 98);
INSERT INTO students VALUES (2, 1, 'Madhu', 97);
INSERT INTO students VALUES (2, 2, 'Om', 95);

INSERT INTO students(name,marks) VALUES ('Rahul', 95);--error
INSERT INTO students(std,name,marks) VALUES (1,'Pratik', 85);--error
INSERT INTO students(std,roll,name,marks) VALUES (1,2,'Onkar', 75);--error
INSERT INTO students VALUES (2, 3, 'Vishaka', 95);
```

Surrogate Primary Key

```
Usually auto-generated.
ORACLE->Sequences
MSSQL -> Identity
MySQL -> Auto_Increment
```

```
CREATE TABLE products(id INT PRIMARY KEY AUTO_INCREMENT, name CHAR(20),price
DECIMAL(9,2));

INSERT INTO products(name,price) VALUES('sugar',35);
INSERT INTO products(name,price) VALUES('Lemon',10);
INSERT INTO products(name,price) VALUES('Tata Salt',25);
INSERT INTO products(name,price) VALUES('Almonds',540);

SELECT * FROM products;

ALTER TABLE PRODUCTS AUTO_INCREMENT=100;

INSERT INTO products(name,price) VALUES('Maggi',15);
INSERT INTO products(name,price) VALUES('Amul Cheese',115);
```

4. Foreign key

```
DROP TABLE emps;
DROP TABLE depts;

CREATE TABLE depts (
deptno INT,
dname VARCHAR(20),
PRIMARY KEY(deptno)
);
INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

CREATE TABLE emps(
empno INT,
ename VARCHAR(20),
deptno INT,
mgr INT,
FOREIGN KEY(deptno) REFERENCES depts(deptno)
);

INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);

INSERT INTO emps VALUES (4, 'Nitin', 50, 5);
--error foreign key constraint fails

INSERT INTO emps VALUES (5, 'Sarang', 50, NULL);
--error foreign key constraint fails

INSERT INTO emps VALUES (4, 'Nitin', 30, 5);
INSERT INTO emps VALUES (5, 'Sarang', 30, NULL);
INSERT INTO emps VALUES (6, 'Rohan', NULL, NULL);


SELECT * FROM emps;
SELECT * FROM depts;

DELETE FROM depts WHERE deptno=40;

DELETE FROM depts WHERE deptno=30;
--error foreign key constarint fails

DROP TABLE depts;
--error foreign key constraint fails

DROP TABLE emps;
DROP TABLE depts;
```

```
CREATE TABLE depts (
deptno INT,
dname VARCHAR(20),
PRIMARY KEY(deptno)
);
INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

CREATE TABLE emps(
empno INT,
ename VARCHAR(20),
deptno INT,
mgr INT,
FOREIGN KEY(deptno)
REFERENCES depts(deptno)
ON DELETE CASCADE ON UPDATE CASCADE
);

INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);
INSERT INTO emps VALUES (4, 'Nitin', 30, 5);
INSERT INTO emps VALUES (5, 'Sarang', 30, NULL);

UPDATE depts SET deptno=50 WHERE deptno=30;
--changes will happen in depts as well as in emps

DELETE FROM depts WHERE deptno=50;
--changes will happen in depts as well as in emps


CREATE TABLE stu(std INT, roll INT, name CHAR(30),PRIMARY KEY(std,roll));

INSERT INTO stu VALUES (1, 1, 'Soham');
INSERT INTO stu VALUES (1, 2, 'Sakshi');
INSERT INTO stu VALUES (1, 3, 'Prisha');
INSERT INTO stu VALUES (2, 1, 'Madhu');
INSERT INTO stu VALUES (2, 2, 'Om');

CREATE TABLE marks(std INT, roll INT, marks DECIMAL(5,2),FOREIGN KEY (std,roll)
REFERENCES stu(std,roll));

INSERT INTO marks VALUES (1, 1, 50);
INSERT INTO marks VALUES (1, 2, 60);
INSERT INTO marks VALUES (1, 3, 70);
INSERT INTO marks VALUES (2, 1, 80);
INSERT INTO marks VALUES (2, 2, 90);

CREATE TABLE depts (
deptno INT,
dname VARCHAR(20),
PRIMARY KEY(deptno)
```

```
);
INSERT INTO depts VALUES (10, 'DEV');
INSERT INTO depts VALUES (20, 'QA');
INSERT INTO depts VALUES (30, 'OPS');
INSERT INTO depts VALUES (40, 'ACC');

CREATE TABLE emps(
empno INT PRIMARY KEY,
ename VARCHAR(20),
deptno INT,
mgr INT,
FOREIGN KEY(deptno) REFERENCES depts(deptno)
);


INSERT INTO emps VALUES (1, 'Amit', 10, 4);
INSERT INTO emps VALUES (2, 'Rahul', 10, 3);
INSERT INTO emps VALUES (3, 'Nilesh', 20, 4);

INSERT INTO emps VALUES (4, 'Nitin', 50, 5);
--cannot insert

SELECT @@foreign_key_checks; --1

SET @@foreign_key_checks=0;

SELECT @@foreign_key_checks; --0

--Now you can insert the data of dept 50 as the foreign key checks are disabled.
INSERT INTO emps VALUES (4, 'Nitin', 50, 5);
INSERT INTO emps VALUES (5, 'Sarang', 50, NULL);

SET @@foreign_key_checks=1;
INSERT INTO emps VALUES (6, 'Rohan', 60, 4);

CREATE TABLE depts_backup (
deptno INT,
dname VARCHAR(20),
PRIMARY KEY(deptno)
);

CREATE TABLE emps_backup(
empno INT PRIMARY KEY,
ename VARCHAR(20),
deptno INT,
mgr INT,
FOREIGN KEY(deptno) REFERENCES depts_backup(deptno)
);

SET @@foreign_key_checks=0;

INSERT INTO emps_backup SELECT empno,ename,deptno,mgr FROM emp;

SET @@foreign_key_checks=1;
```

```
--self referencing Foreign Key
CREATE TABLE emps2(
empno INT PRIMARY KEY,
ename VARCHAR(20),
deptno INT,
mgr INT,
FOREIGN KEY(deptno) REFERENCES depts(deptno),
FOREIGN KEY(mgr) REFERENCES emps2(empno)
);
```

5. CHECK

```
CREATE TABLE voters(
name CHAR(20),
age INT NOT NULL CHECK (age>18)
);

INSERT INTO voters VALUES('Rohan',28);--OK
INSERT INTO voters VALUES('Ranbir',16);--NOT OK
INSERT INTO voters VALUES('Alia',NULL);--NOT OK
```

Contraint names

```
SHOW CREATE TABLE emps;

--ColOUMN LEVEL CONSTRAINT
CREATE TABLE customer(
adharcard CHAR(12) PRIMARY KEY,
name CHAR(30) NOT NULL,
email VARCHAR(50) UNIQUE,
password CHAR(20) NOT NULL CHECK(LENGTH(password)>8),
mobile CHAR(10) UNIQUE NOT NULL,
age INT NOT NULL
);

CREATE TABLE ADDRESS(
id INT PRIMARY KEY AUTO_INCREMENT,
adharcard CHAR(12),
buildingno CHAR(5),
city CHAR(15) NOT NULL,
pin INT NOT NULL,
landmark VARCHAR(20),
FOREIGN KEY adharcard REFERENCES customer(adharcard)
);


CREATE TABLE customer(
```

```
adharcard CHAR(12),
name CHAR(30) NOT NULL,
email VARCHAR(50),
password CHAR(20) NOT NULL,
mobile CHAR(10) NOT NULL,
age INT NOT NULL,
CONSTRAINT pk_adharcard PRIMARY KEY(adharcard),
CONSTRAINT un_email UNIQUE(email),
CONSTRAINT ch_password CHECK(LENGTH(password)>8),
CONSTRAINT un_mobile UNIQUE(mobile)
);

CREATE TABLE ADDRESS(
id INT PRIMARY KEY AUTO_INCREMENT,
adharcard CHAR(12),
buildingno CHAR(5),
city CHAR(15) NOT NULL,
pin INT NOT NULL,
landmark VARCHAR(20),
CONSTRAINT fk_adharcard FOREIGN KEY adharcard REFERENCES customer(adharcard)
);
```

ALTER

```
- Alter is used to modify the structure of table
- Not recommended when database is in production level
- After alter you table structure becomes inefficient
```

```
DESC emps;

--add a column job in the emps table
ALTER TABLE emps ADD COLUMN job VARCHAR(20);
DESC emps;
SELECT * FROM emps;

--change the datatype of job
ALTER TABLE emps MODIFY job VARCHAR(30);
DESC emps;

--change name of column
ALTER TABLE emps CHANGE ename name varchar(20);
DESC emps;

--Remove a column job
ALTER TABLE emps DROP COLUMN job;
DESC emps;

--Add a constraint
ALTER TABLE emps ADD CONSTRAINT UNIQUE(name);
```

```
DESC emps;

--DELETE a constraint
ALTER TABLE emps DROP CONSTRAINT name;
ALTER TABLE emps DROP CONSTRAINT emps_ibfk_1;
DESC emps;

--rename a table
```

AGENDA

```
PL/SQL
Exception Handling
Cursor
Functions
```

PL/SQL

```
Procedural Language
PSM - Persistant Stored Modules
```

Delimiter

```
Default delimiter is  ; in mysql
When ; is found the code is submitted to mysql server and it is processed.
It needs to be changed temporarily for the Stored Procedure using DELIMITER
keyword.
```

```
DELIMITER $$
SELECT * FROM emp; -- Wil not execute
$$ -- the statement will now be executed

DELIMITER ; -- Changing back the delimiter
```

Steps for stored Procedure Programming

```
1. Create a file with extension as .sql
2. Write the procedure follwing the proper syntax
3. Save the File
4. use SOURCE cmd to import the file in mysql
5. CALL the procedure
```

To look for all procedures use below query

```
 SHOW PROCEDURE STATUS WHERE db='classwork';
```

```
CREATE TABLE result(id INT, msg CHAR(20));
```

Errors in Mysql

```
1. Error code
    1045 - Access denied
    1062 - Duplicate entry
    1146 - Table doesn't exist

2. Error State
    28000 - Access denied
    23000 - Duplicate entry
    42S02 - Table doesn't exist
```

Exception Handling

```
CREATE PROCEDURE sp_placeorder()
BEGIN

DECLARE EXIT HANDLER FOR error
BEGIN
ROLLBACK;
SELECT 'Order Failed - Try Again' AS msg;
END;

START TRANSACTION;
INSERT INTO orders VALUES(...);
INSERT INTO orders_items VALUES(...);
INSERT INTO payment VALUES(...);
COMMIT;

END;
```

CURSOR

Cursor is a special variable in PSM used to access rows/values "one by one" from result of "SELECT" statement.

Programming Steps

1. Declare handler for end of cursor (like end-of-file). Error code: "NOT FOUND".
2. Declare cursor variable with its SELECT statement.
3. Open cursor.
4. Fetch (current row) values from cursor into some variables & process them.
5. Repeat process all rows in SELECT output. At the end error handler will be executed.
6. Exit the loop and close the cursor.

Characteristics of "MySQL Cursors"

1. Readonly
    - We can use cursor only for reading from the table.
    - Cannot update or delete from the cursor.
    - SET v_cur1 = (1, 'NewName'); -- not allowed
    - To update or delete programmer can use UPDATE/DELETE queries.
2. Non-scrollable
    - Cursor is forward only.
    - Reverse traversal or random access of rows is not supported.
    - When FETCH is done, current row is accessed and cursor automatically go to next row.
    - We can close cursor and reopen it. Now it again start iterating from the start.
3. Asensitive
    - When cursor is opened, the addresses of all rows (as per SELECT query) are recorded into the cursor (internally). These rows are accessed one by one (using FETCH).
    - While cursor is in use, if other client modify any of the rows, then cursor get modified values.
    - Because cursor is only having address of rows, Cursor is not creating copy of the rows. Hence MySQL cursors are faster.

Functions

- User-defined Functions

1. DETERMINISTIC
    - If input is same, output will remain same ALWAYS.
    - Internally MySQL cache input values and corresponding output.
    - If same input is given again, directly output may return to speedup execution.
2. NOT DETERMINISTIC
    - Even if input is same, output may differ.
    - Output also depend on current date-time or state of table or database settings.
    - These functions cannot be speedup.
    - Before importing NOT Deterministic Function do the below Setting.
    - SET GLOBAL @@log_bin_trust_function_creators=1;

Agenda

```
Triggers
Codd's Rule
Normalization
Introduction to Mongo
```

Triggers

```
Triggers in MySQL are Programs(PSM Syntax).
It is executed(triggered) by some event.
DML Operations
AFTER INSERT
AFTER UPDATE
AFTER DELETE
BEFORE INSERT
BEFORE UPDATE
BEFORE DELETE

IF DML operation is done on multiple rows then for each individual row a trigger
will be fired.

NEW and OLD are the keywords.
These keywords are used on affected rows.
INSERT -> NEW
DELETE -> OLD
UPDATE -> NEW ,OLD

Triggers cannot be called explicitly by user.
```

```sql
DROP TABLE accounts;
CREATE TABLE accounts(
id INT,
type CHAR(15),
balance DECIMAL(10,2)
);

INSERT INTO accounts VALUES(1,'SAVINGS',0);
INSERT INTO accounts VALUES(2,'SAVINGS',0);
INSERT INTO accounts VALUES(3,'CURRENT',0);

CREATE TABLE transactions(
    tid INT PRIMARY KEY AUTO_INCREMENT,
    type CHAR(15),
    acc_id INT,
```

```
    amount DECIMAL(10,2)
);

INSERT INTO transactions(type,acc_id,amount) VALUES('Deposit',1,5000);
INSERT INTO transactions(type,acc_id,amount) VALUES('Deposit',2,8000);
INSERT INTO transactions(type,acc_id,amount) VALUES('Deposit',3,10000);

INSERT INTO transactions(type,acc_id,amount) VALUES('Withdraw',2,2000);
```

Stored Procedure – PSM Syntax

VARIABLES

```
DECLARE varname DATATYPE;
DECLARE varname DATATYPE DEFAULT init_value;
SET varname = new_value;
SELECT new_value INTO varname;
SELECT expr_or_col INTO varname FROM table_name;
```

PARAMETERS

```
CREATE PROCEDURE sp_name(PARAMTYPE p1 DATATYPE)
BEGIN
    …
END;

-- IN param: Initialized by calling program.
-- OUT param: Initialized by called procedure.
-- INOUT param: Initialized by calling program and
modified by called procedure
-- OUT & INOUT param declared as session variables.

CREATE PROCEDURE sp_name(OUT p1 INT)
BEGIN
    SELECT 1 INTO p1;
END;

SET @res = 0;
CALL sp_name(@res);
SELECT @res;
```

IF-ELSE

```
IF condition THEN
     body;
END IF;
-----------------------
IF condition THEN
     if-body;
ELSE
     else-body;
END IF;
-----------------------
IF condition THEN
     if1-body;
ELSE
     IF condition THEN
            if2-body;
     ELSE
            else2-body;
     END IF;
END IF;
-----------------------
IF condition THEN
     if1-body;
ELSEIF condition THEN
     if2-body;
ELSE
     else-body;
END IF;
```

LOOPS

```
WHILE condition DO
    body;
END WHILE;
------------------
REPEAT
    body;
UNTIL condition
END REPEAT;
------------------
label: LOOP
IF condition THEN
        ...
        LEAVE label;
    END IF;
    ...
END LOOP;
```

CASE-WHEN

```
CASE
WHEN condition THEN
      body;
WHEN condition THEN
      body;
ELSE
       body;
END CASE;
```

SHOW PROCEDURE

```
SHOW PROCEDURE STATUS
LIKE ‘sp_name’;

SHOW CREATE PROCEDURE sp_name;
```

DROP PROCEDURE

```
DROP PROCEDURE
IF EXISTS sp_name;
```