select deptno, job,ename,sal, hiredate from emp

order by deptno, job;

```
SQL> select deptno, job, ename, sal, hiredate from emp
  2  order by deptno, job;

    DEPTNO JOB       ENAME             SAL HIREDATE
---------- --------- ---------- ---------- ---------
        10 CLERK     MILLER           1300 23-JAN-82
        10 MANAGER   CLARK            2450 09-JUN-81
        10 PRESIDENT KING             5000 17-NOV-81
        20 ANALYST   FORD             3000 03-DEC-81
        20 ANALYST   SCOTT            3000 09-DEC-82
        20 CLERK     ADAMS            1100 12-JAN-83
        20 CLERK     SMITH             800 17-DEC-80
        20 MANAGER   JONES            2975 02-APR-81
        30 CLERK     JAMES             950 03-DEC-81
        30 MANAGER   BLAKE            2850 01-MAY-81
        30 SALESMAN  WARD             1250 22-FEB-81
        30 SALESMAN  ALLEN            1600 20-FEB-81
        30 SALESMAN  MARTIN           1250 28-SEP-81
        30 SALESMAN  TURNER           1500 08-SEP-81

14 rows selected.
```

select deptno, job,ename,sal, hiredate from emp

order by deptno desc, job;

select deptno, job, ename, sal, hiredate from emp

order by deptno, job desc;

- No upper limit on the number of columns in order by clause
- Select …….
- Order by country, state, district,city;
- Separated by commas

- If you have large number of rows in the table,and if you have large number of columns in ORDER by clause, then your SELECT statement will be slow; because that much sorting has to take place in server ram
- Sorting is one operation that always slows down your select statement

Select ename, sal*12 from emp;

```
SQL> select ename, sal*12 from emp;

ENAME          SAL*12
---------- ----------
KING            60000
BLAKE           34200
CLARK           29400
JONES           35700
MARTIN          15000
ALLEN           19200
TURNER          18000
JAMES           11400
WARD            15000
FORD            36000
SMITH            9600
SCOTT           36000
ADAMS           13200
MILLER          15600

14 rows selected.
```

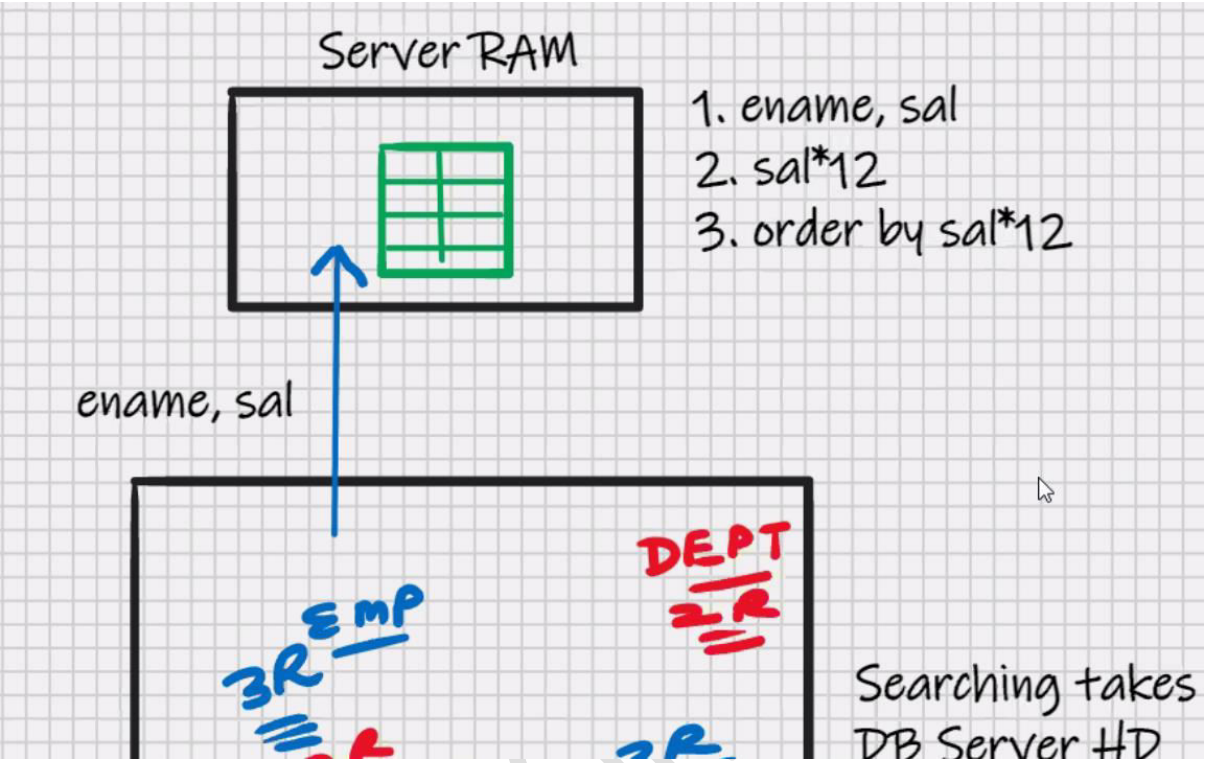
Select ename, sal*12 from emp;

Order by sal*12;

```
SQL> select ename, sal*12 from emp
  2  order by sal*12;

ENAME          SAL*12
---------- ----------
SMITH            9600
JAMES           11400
ADAMS           13200
WARD            15000
MARTIN          15000
MILLER          15600
TURNER          18000
ALLEN           19200
CLARK           29400
BLAKE           34200
JONES           35700
FORD            36000
SCOTT           36000
KING            60000

14 rows selected.
```

Select ename, sal*12 from emp;

Order by sal*12 desc;

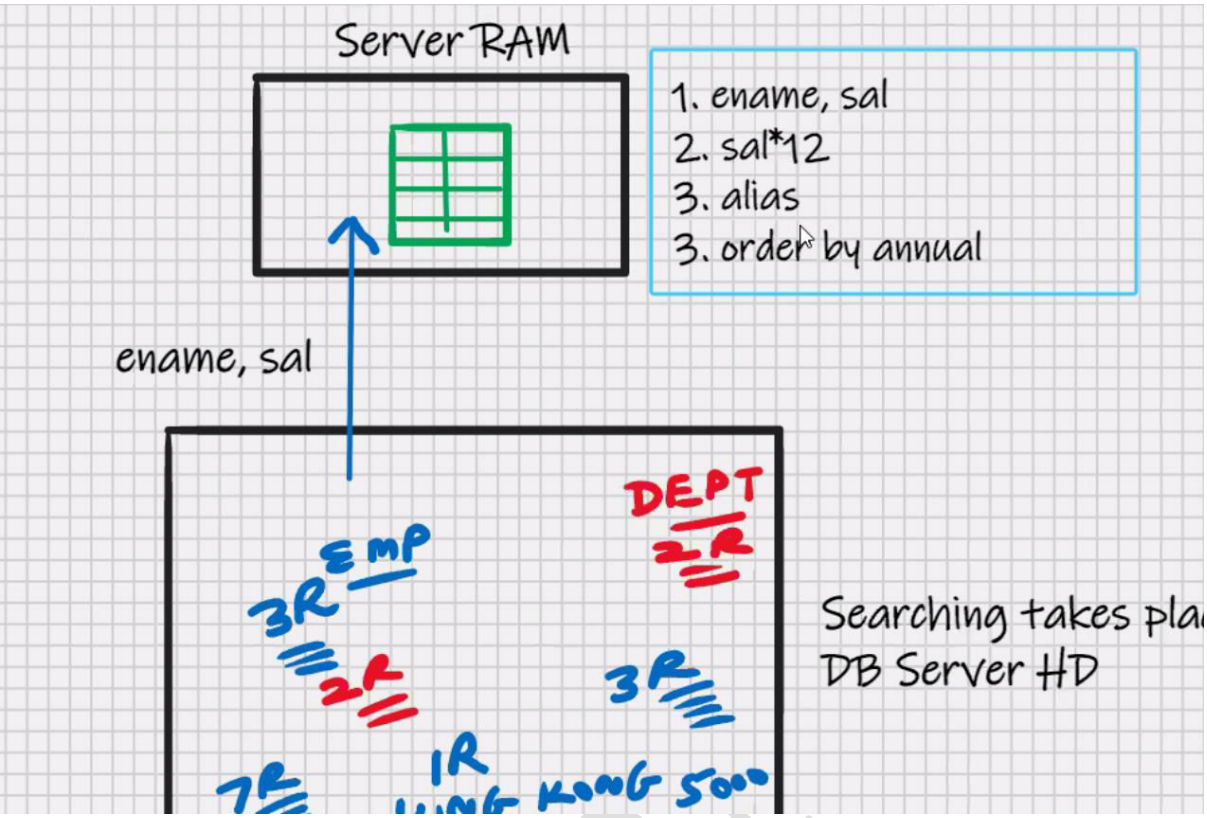


Select ename, sal*12 annual from emp;

Order by annual;

Select ename, sal*12 “annual” from emp;

Order by “annual”;



Select ename, sal*12 “annual salary” from emp;

Order by 2;

```
SQL> select ename, sal*12 "Annual Salary" from emp
  2  order by 2;

ENAME      Annual Salary
---------- -------------
SMITH                9600
JAMES               11400
ADAMS               13200
WARD                15000
MARTIN              15000
MILLER              15600
TURNER              18000
ALLEN               19200
CLARK               29400
BLAKE               34200
JONES               35700
FORD                36000
SCOTT               36000
KING                60000

14 rows selected.

SQL>
```

Select * from emp

Order by 2;

MySQL – SQL-Special operator (like,between,any)

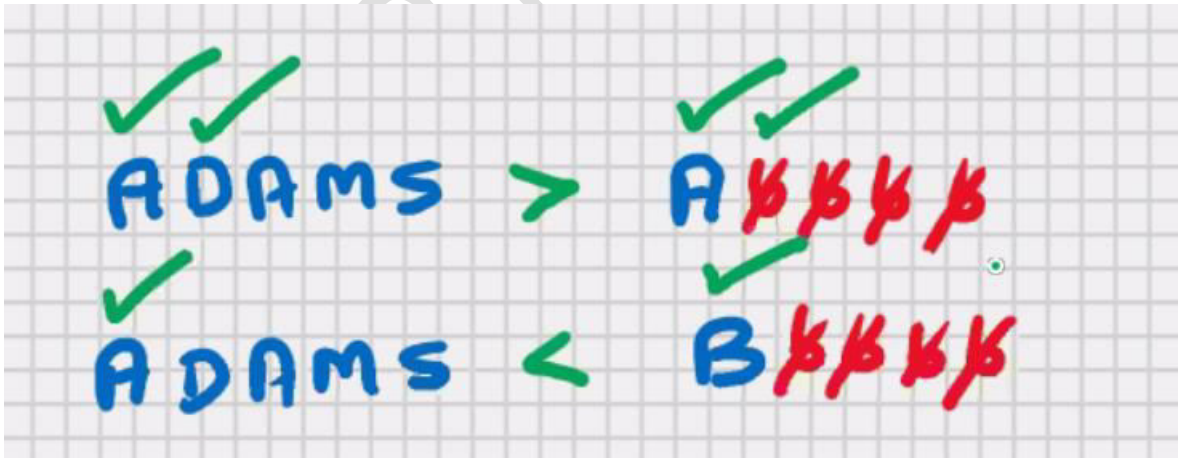
EMP

EMPNO	ENAEM	SAL	CITY	DEPTNO
1	ADAMS	1000	MUMBAI	10
2	BLAKE	2000	DELHI	10
3	ALLEN	2500	MUMBAI	20
4	KING	3000	DELHI	25
5	FORD	4000	BHOPAL	30

Select * from emp

Where ename > ‘A’ and ename < ‘B’ ;

```
SQL> select * from emp
  2  where ename > 'A' and ename < 'B';

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7499 ALLEN      SALESMAN         7698 20-FEB-81       1600        300         30
      7876 ADAMS      CLERK            7788 12-JAN-83       1100                    20

SQL>
```



Blank-padded comparision semansistics:-

When you compare two strings of different lengths, the shorter of the 2 strings is temporarily padded with blank spaces on RHS, such that their lengths become equal; then it will start the comparisons, character by character, based on ASCII value

```
Select * from emp
Where ename >= 'A' and ename < 'B' ;

Select * from emp
Where ename like 'A%';
```

Wildcards(used for pattern matching)

%	any character and any number of characters
_	any 1 character

To make it case insensitive , solution for oracle:-

```
Select * from emp
Where ename like 'A%' or ename like 'a%';

Select * from emp
Where ename like '%A';  //ending with A

Select * from emp
Where ename like '%A%';   //any where with A

Select * from emp
Where ename like '_ _A%';  //all rows with A as third letter

Select * from emp
Where ename like '_ _ _ _';      //king   ford

Select * from emp
Where ename like 'A _ _ B';

Select * from emp
Where ename like 'A%';
```

Select * from emp
Where ename not like ‘A%’; //not starting with A

Select * from emp
Where sal >= 2000 and sal <= 3000;

		EMP		
EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	1000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Select * from emp
Where sal between 2000 and 3000; // mutually inclusive
**recommended*
Easier to write and works faster

		EMP		
EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	1000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Select * from emp
Where sal not between 2000 and 3000; //exclusive

		EMP		
EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	1000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Select * from emp
Where hiredate >= '2021-01-01' and hiredate <= '2021-12-31';

Select * from emp
Where hiredate between '2021-01-01' and '2021-12-31';
**Recommended*

Select * from emp
Where ename >= 'A' and ename <= 'F';

Select * from emp
Where ename between 'A' and 'F';
**Recommended*

Select * from emp
Where ename like 'A%' or ename like 'F%';

Special operator – Any,in

```
Select * from emp
Where deptno=10 or deptno=20 or deptno=30;

Select * from emp
Where deptno=any(10,20,30);
Select * from emp
Where deptno (=,>,>=,<,<=,!=,==) any(10,20,30);
```

EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	1000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

- **Any** operator will perform **Logical OR**
- Easier to write and Works faster

```
Select * from emp
Where deptno in(10,20,30);
```

- **in** operator will perform **Logical OR**
- Easier to write and Works faster(**Fastest**)

```
Select * from emp
Where deptno not in(10,20,30);
```

- In operator is faster than any operator

- Any operator is more powerful than in operator
- With in operator, you can only check for **in and not in**
- With any operator you can check for = any, !=any, >any, >=any, <=any
- If you want to check for equality or inequality, then use the in operator if you want to check for >,>=,<,<= then use the **any** operator
- In and any operator works with **all data types**.

Select * from emp
Where city in('mumbai','bhopal');

Select * from emp
Where city not in('mumbai','bhopal');

Select * from emp
Where city = any('mumbai','bhopal');

- Any operator does not work directly in MySQL e.g.

Select * from emp
Where deptno= any (10,20);   //not supported in mysql

- In mysql , **any** operator has to be used with **sub-query**
- In mysql, **use the in operator**

MySQL – SQL

DDL -> create, drop
DML -> insert , update , delete
DQL -> select

UPDATE

Update emp
Set sal = 10000
Where empno = 1;

EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	10000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Update emp
Set sal = sal + sal * 0.4
Where empno = 1; //increased by 40%

Update emp

Set sal = 10000, city = ‘nashik’

Where empno = 1;

EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	~~1000~~ 10000	~~Mumbai~~ Nashik	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Update emp

Set sal = 10000

Where city =’mumbai’;

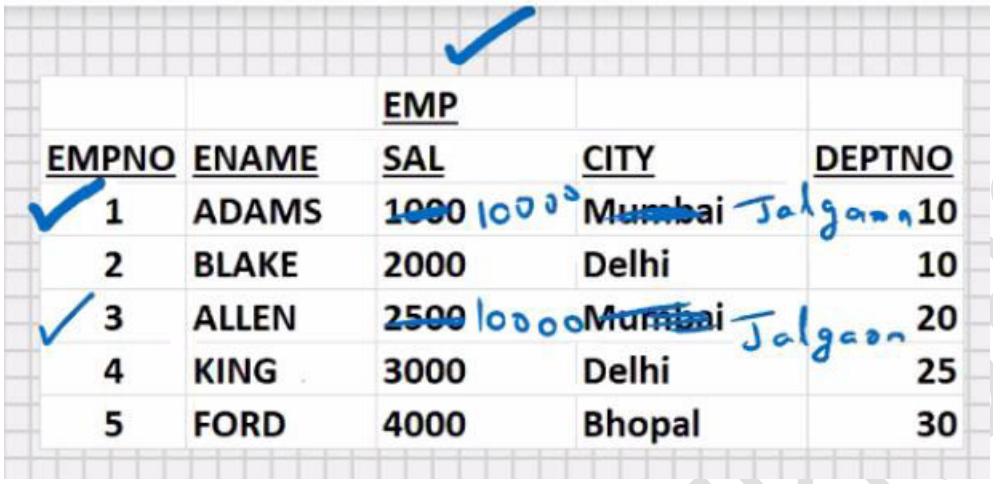
EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	~~1000~~ 10000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	~~2500~~ 10000	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Update emp

Set sal = 10000 , city = ‘jalgaon’

Where city =’mumbai’;



EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	1000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

- You can update multiple rows and multiple columns simultaneously, but **only 1 table at a time**
- If you want to update 2 or more tables, then a separate update command is required for each table

Update emp

Set sal = 10000 , city = ‘jalgaon’

Where empno between 1 to 3;

Update emp

Set sal = 10000 ; // all rows will get updated

EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	1000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

DELETE

Delete from emp

Where empno = 1;

EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
~~1~~	~~ADAMS~~	~~1000~~	~~Mumbai~~	~~10~~
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Delete from emp

Where city = ‘mumbai’;

EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
~~1~~	~~ADAMS~~	~~1000~~	~~Mumbai~~	~~10~~
2	BLAKE	2000	Delhi	10
~~3~~	~~ALLEN~~	~~2500~~	~~Mumbai~~	~~20~~
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Delete from emp;

EMP

EMPNO	ENAME	SAL	CITY	DEPTNO
1	ADAMS	1000	Mumbai	10
2	BLAKE	2000	Delhi	10
3	ALLEN	2500	Mumbai	20
4	KING	3000	Delhi	25
5	FORD	4000	Bhopal	30

Drop table emp; // delete the table forever

Drop table emp, dept;   //deleting multiple tables

- You cannot use where clause with drop table,
  Because drop table is a DDL command
- Update and delete commands without where clause will not be allowed in mysql workbench

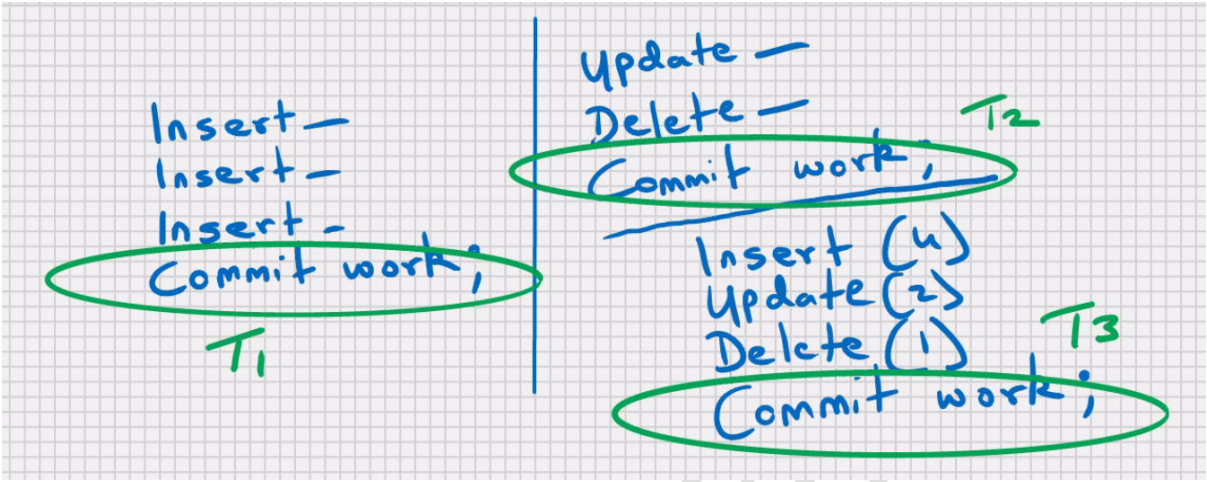
To use update and delete without **where clause in MySql** workbench:-
Click on edit (menu at the top) ->preferences -> SQL editor ->

“safe updates” (checkbox at the bottom) -> uncheck it -> click on OK

This requires a reconnection to the server
Click on Query (menu at the top) -> Reconnect to Server -> click on it

MySQL- SQL – Transaction Processing

- Commit will save all the DML(insert,update,delete) changes since the last committed state



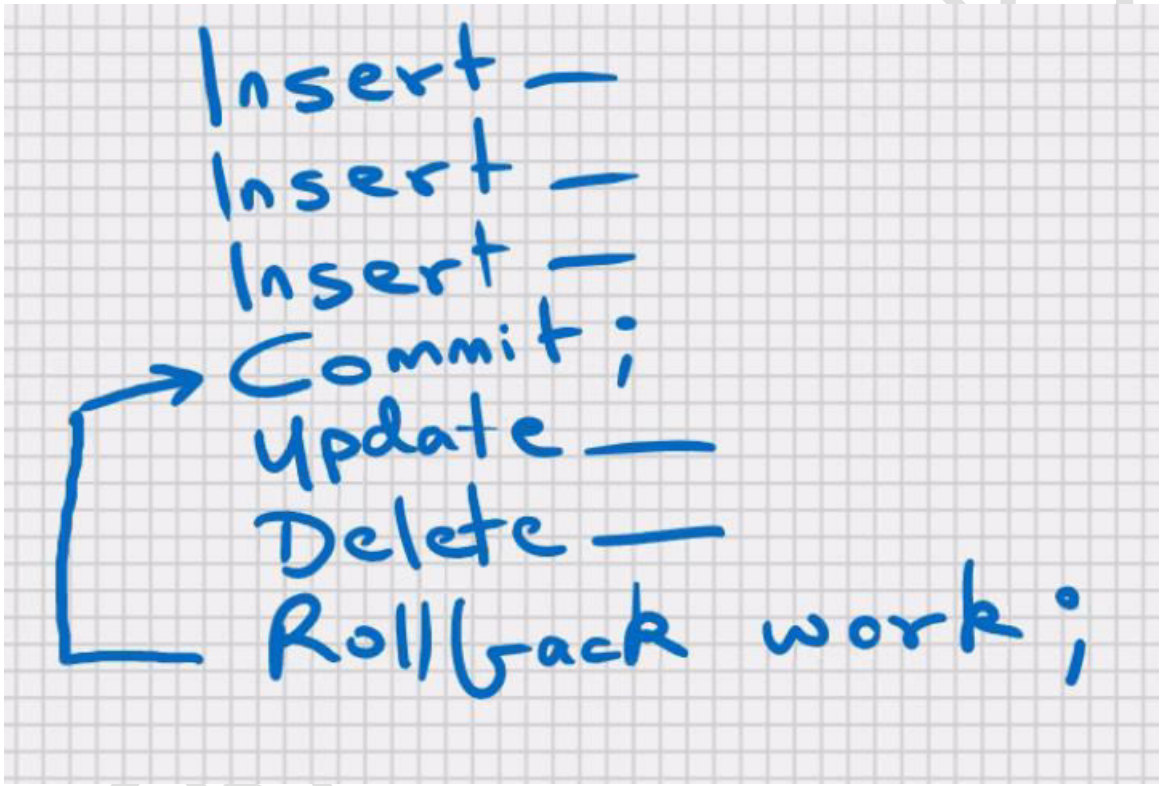
- When the user issues a commit, it is known as end of Transaction
- Commit will make the Transaction permanent

Commit work;

Work -> ANSI SQL
Work -> optional in MySQL and Oracle

Commit;

Total Work Done = T1 + T2 + T3 + … + Tn;

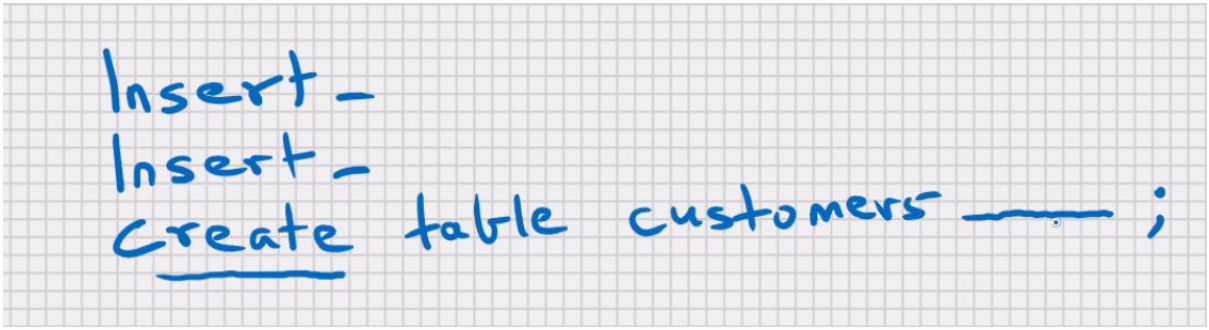


- Rollback will undo all the DML changes since the last commited state what is committed, that cannot be rolled back

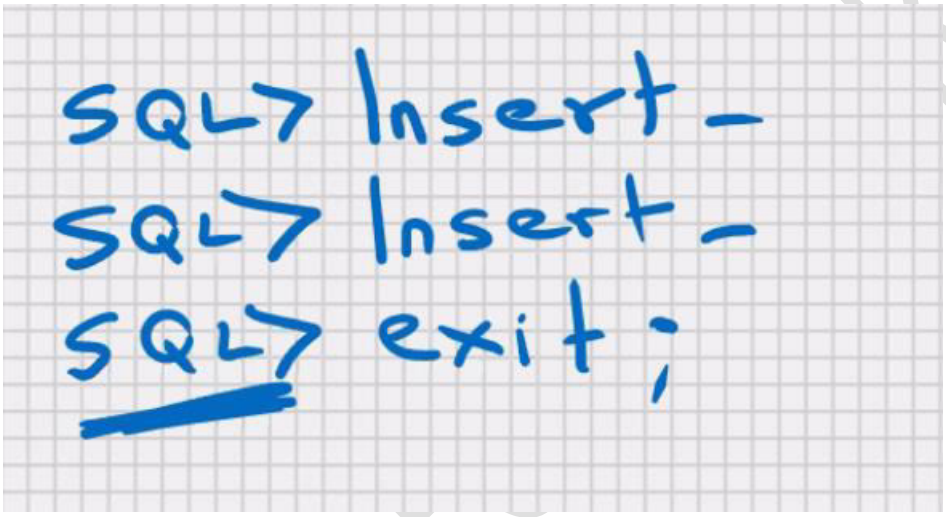
Rollback work;
Or
Rollback

- Work is ANSI SQL
- Work is optional in MySQL and Oracle
- Only the DML commands are affected by rollback and commit

- **Any DDL command, it Automatically commits**



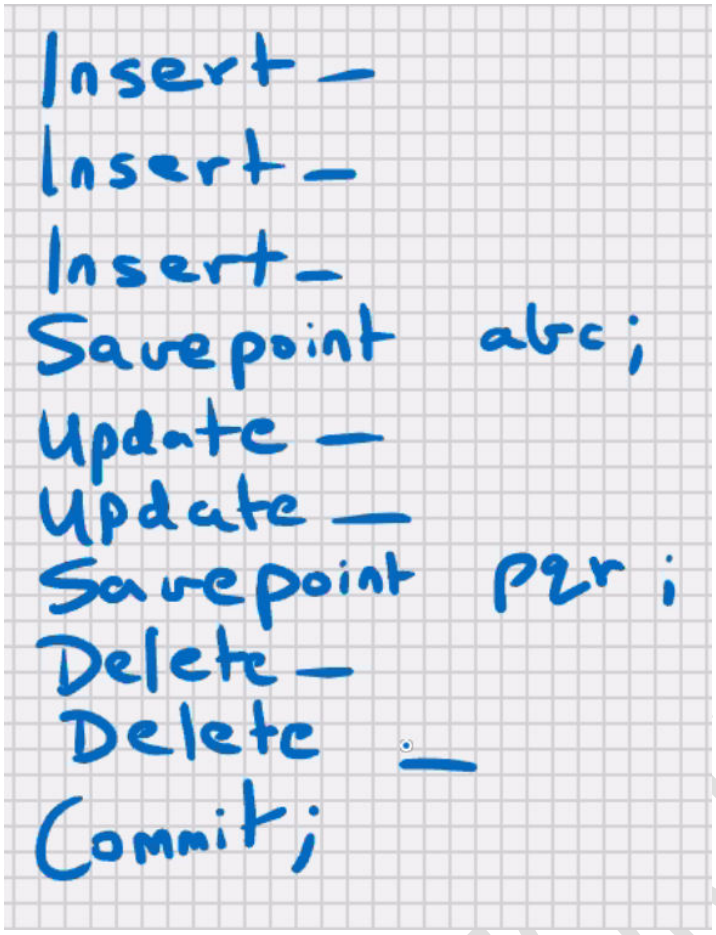
- **In oracle** when you exit from **SQL plus** it automatically commits

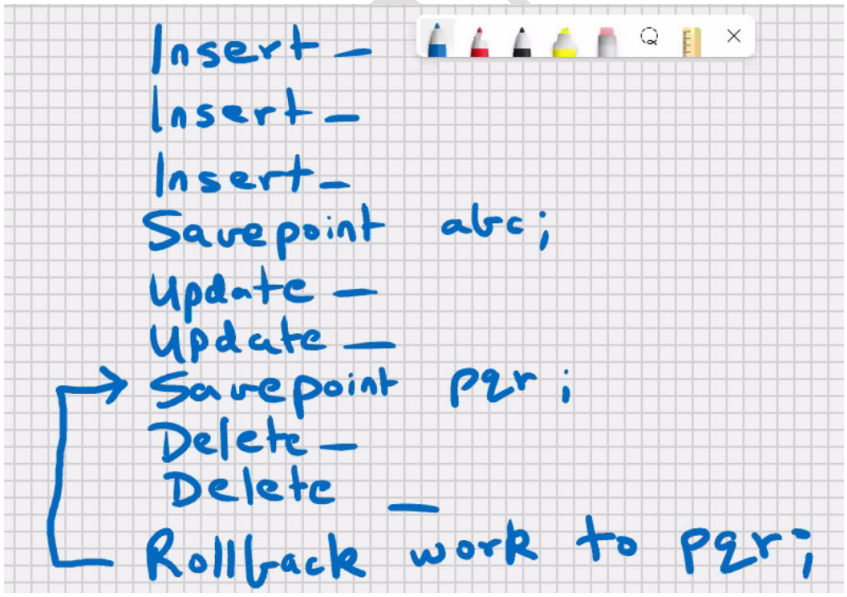


- Any kind of power failure, network failure, system failure, pc reboot, window close , improper exit, etc.; your last uncommitted transaction is automatically rolled back

  To try out rollback, commit, and savepoint in mysql workbench
  Click on query (menu at the top) ->auto commit transactions -> uncheck it

  Assignment
  Oracle 1-4
  SQL 1-2


Insert –
Insert –
Insert –
Savepoint abc;
Update –
Update –
Savepoint pzr;
Delete –
Delete –
Commit;

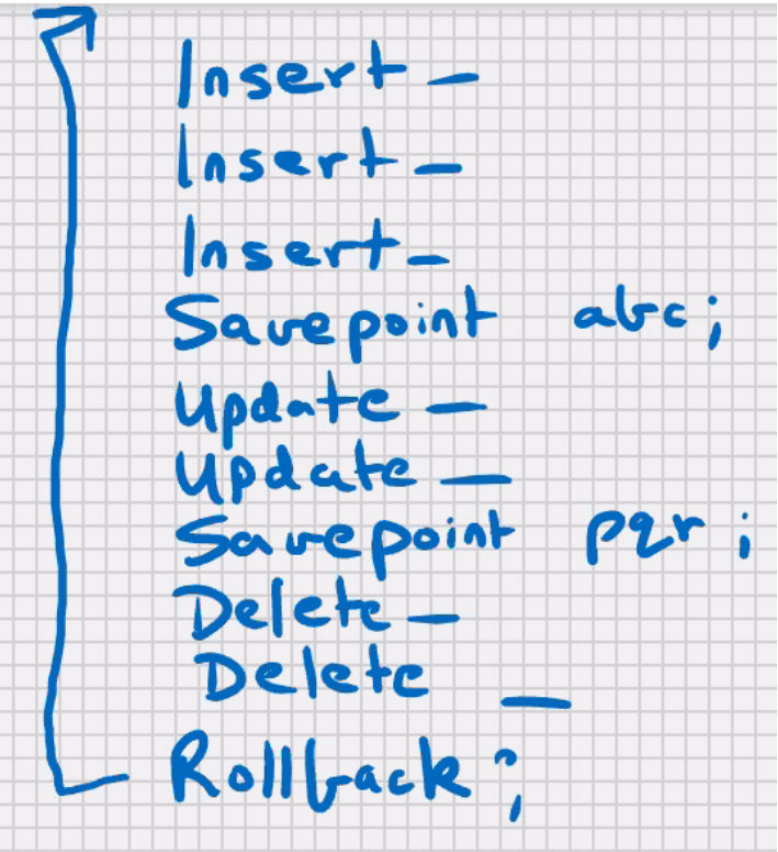

Insert –
Insert –
Insert –
Savepoint abc;
Update –
Update –
Savepoint pzr;
Delete –
Delete –
Rollback work to pzr;

- Savepoint is a point within your work
- Savepoint is a subunit of transaction
- Savepoint is similar to a bookmark
- Within a transaction you can issue the savepoint at regular intervals
- You can rollback to a savepoint

Rollback work to pqr;

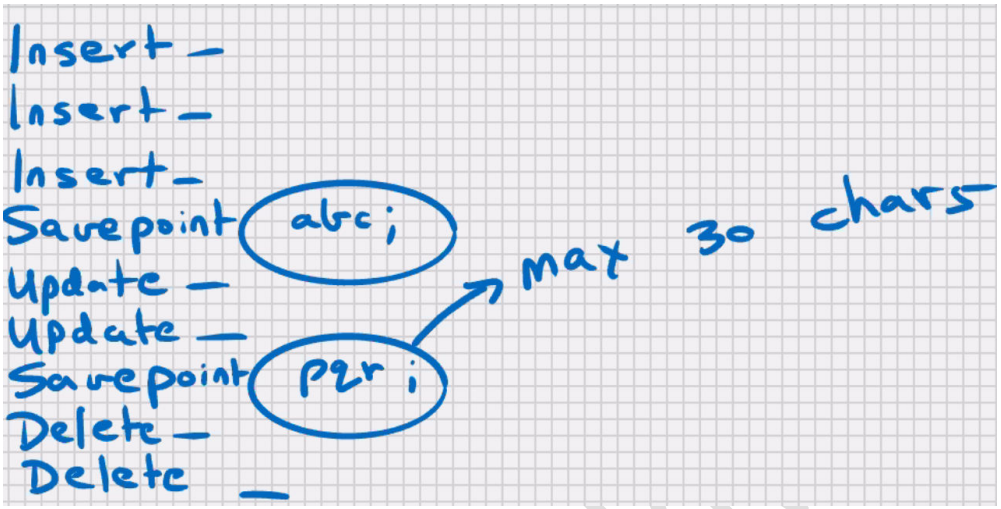
Or
Rollback to pqr;

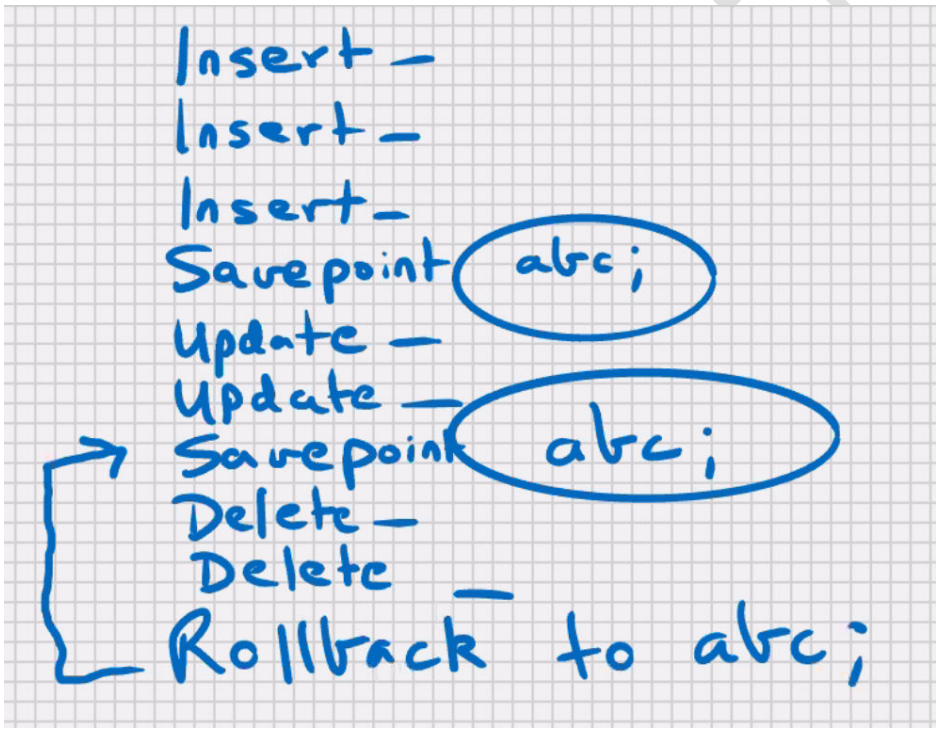
- Work is ANSI SQL
- Work is optional in MySQL and Oracle
- You cannot commit to a savepoint
- Commit will save all the DML changes since the last committed state



- You can only rollback sequentially

- When you rollback or commit, the intermediate savepoints are cleared. If you want to use them again then you will have to reissue them in some new Work

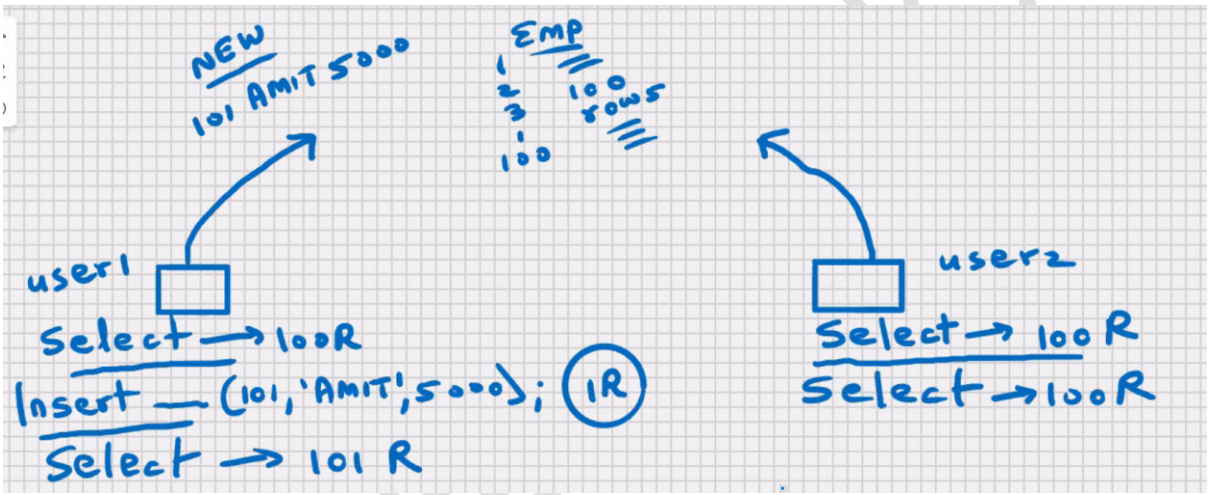


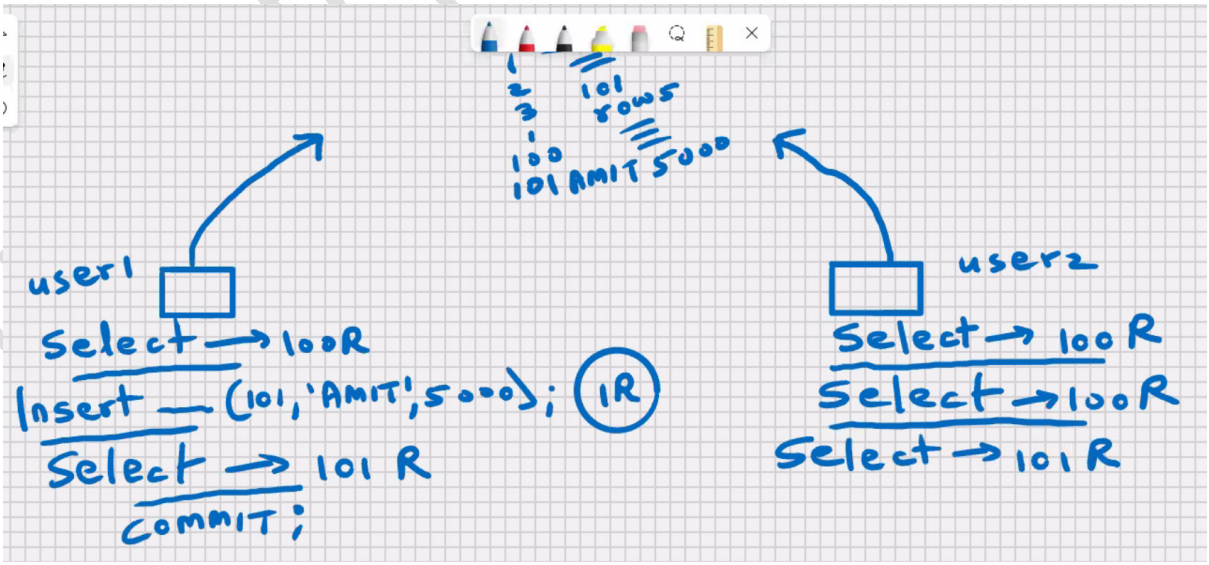


- Within a transaction you can have 2 savepoints with **same name**; the new savepoint **overwrites** the previous one; The older savepoint no longer exists.
- Savepoints will be used in small and large transactions.

- To try out commit, rollback, and savepoint in my sql workbench
- Click on edit (menu at top)->auto commit transactions-> uncheck it

Read and write consistency



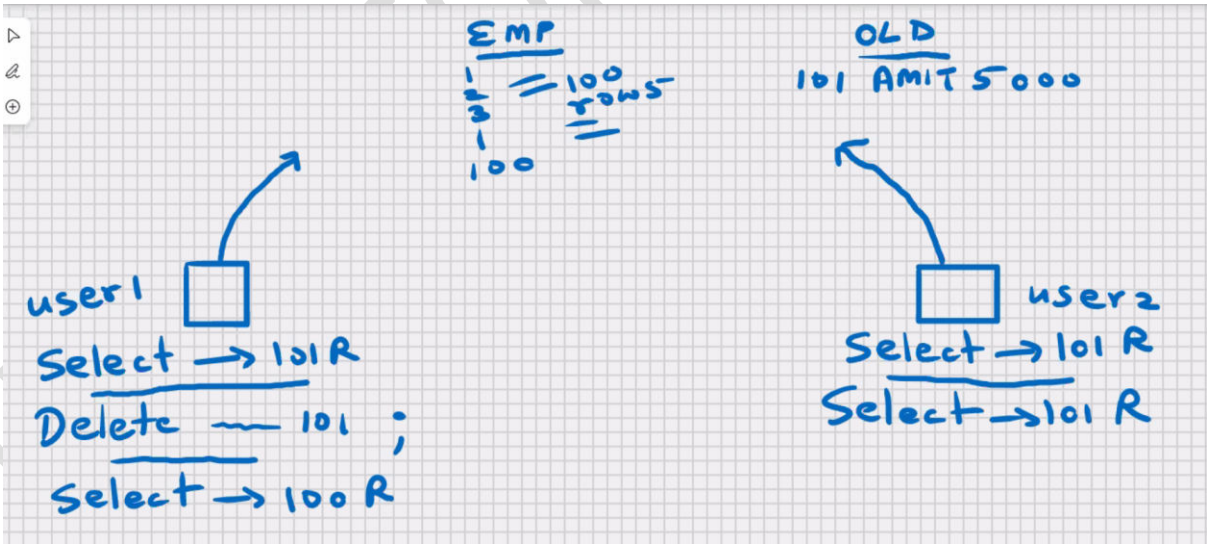


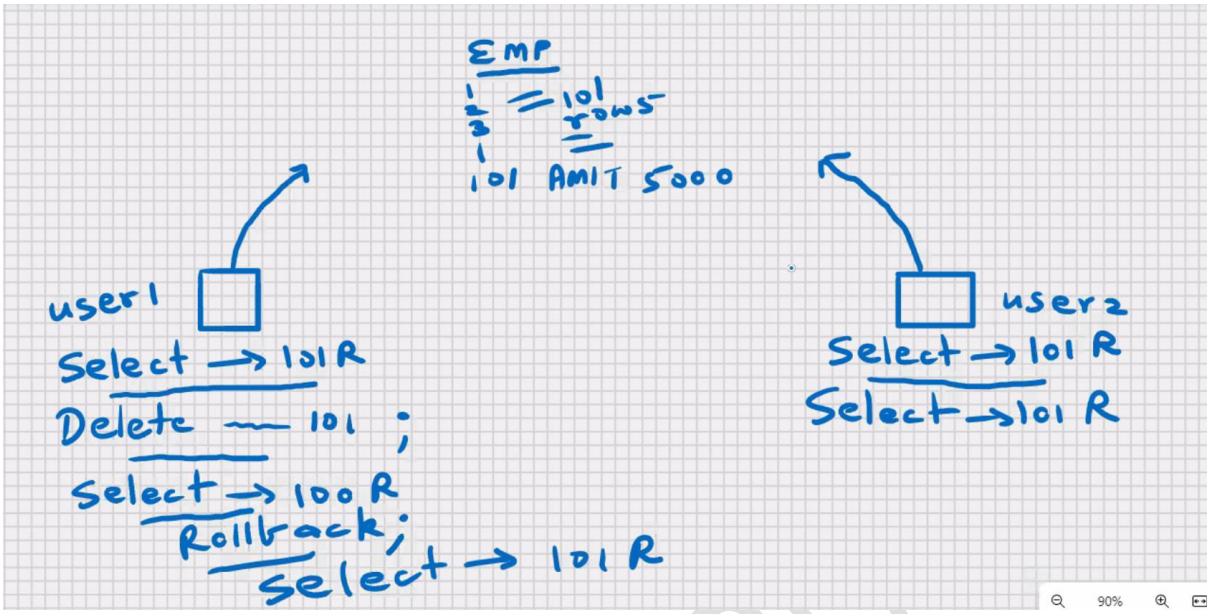
When you select from table you can view only the committed data of all users plus changes made by you

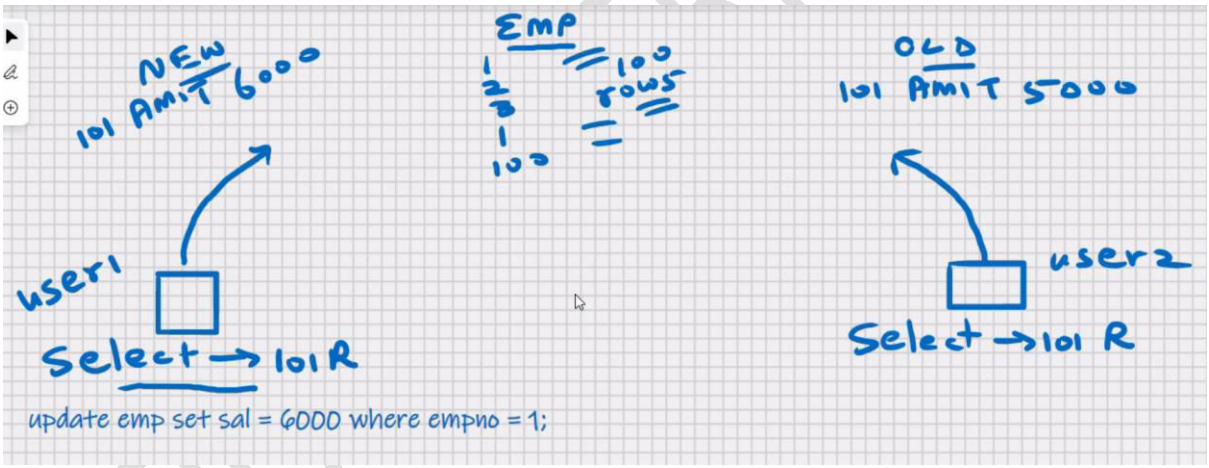
Multiuser environment

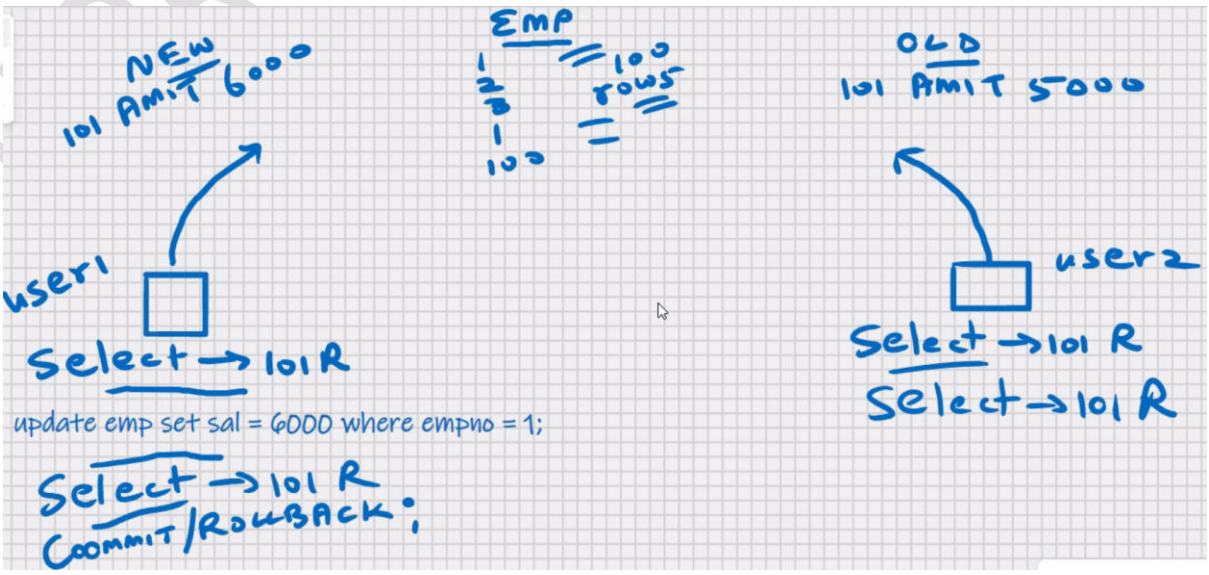
```
USER1> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

USER1> insert into dept values(50, 'TRAINING', 'CDAC');

1 row created.

USER1> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        50 TRAINING       CDAC

USER1> commit;

Commit complete.

USER1>
```

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

USER2> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

USER2> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        50 TRAINING       CDAC

USER2>
```

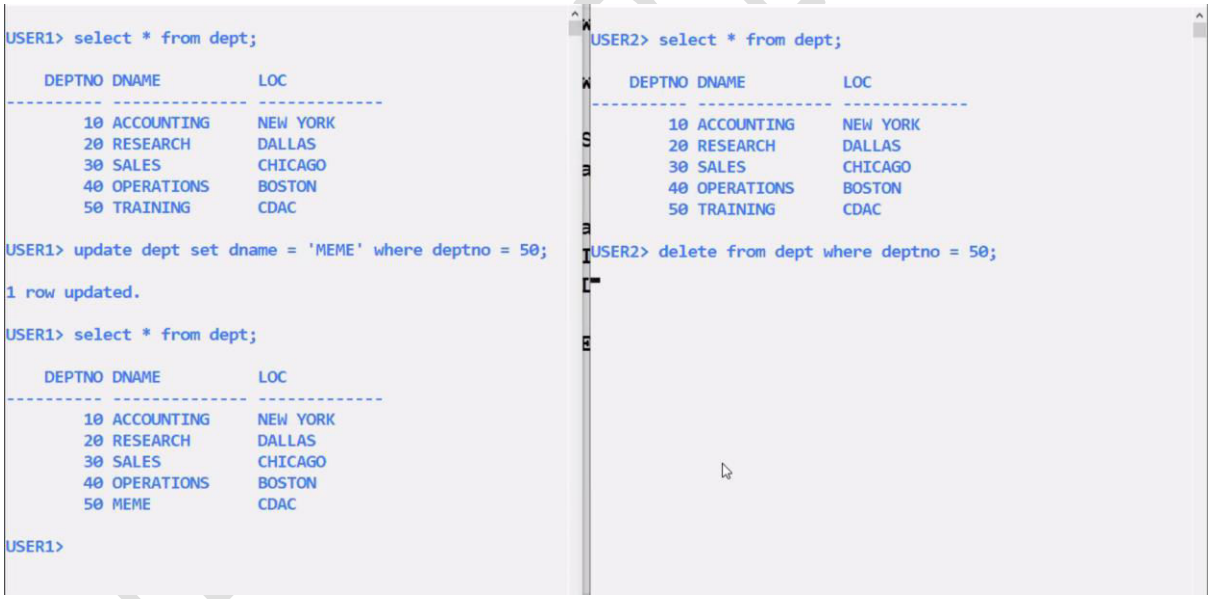



EMP
101 rows
101 AMIT 5000
user1
Select → 101 R
Delete — 101 ;
Select → 100 R
Rollback;
Select → 101 R
user2
Select → 101 R
Select → 101 R
90%


NEW
101 AMIT 6000
EMP
100 rows
OLD
101 AMIT 5000
user1
Select → 101 R
user2
Select → 101 R
update emp set sal = 6000 where empno = 1;


NEW
101 AMIT 6000
EMP
100 rows
OLD
101 AMIT 5000
user1
Select → 101 R
user2
Select → 101 R
Select → 101 R
update emp set sal = 6000 where empno = 1;
Select → 101 R
COMMIT/ROLLBACK;

- When you update or delete a row, that row is automatically locked for users
- When you update or delete a row, that row becomes read_only for other users
- Row locking is automatic in mysql and oracle
- Other users can select from that table; they will view the old data before your changes
- Other users can insert into that table
- Other users can update or delete “other” rows from that table
- No other user can update or delete your locked row, till you have issued a rollback or commit
- Locks are automatically released when you rollback or commit



```
USER1> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        50 MEME           CDAC

USER1> commit;

Commit complete.

USER1>
```

```
USER2> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        50 TRAINING       CDAC

USER2> delete from dept where deptno = 50;

1 row deleted.

USER2> commit;

Commit complete.

USER2>
```

Optimistic locking->automatic row locking mechanism in MySQL and Oracle.

Pessimistic locking->lock the rows Manually in advance before issuing update or delete

- To lock the rows manually, then you have to use select statement with the **for update clause.**

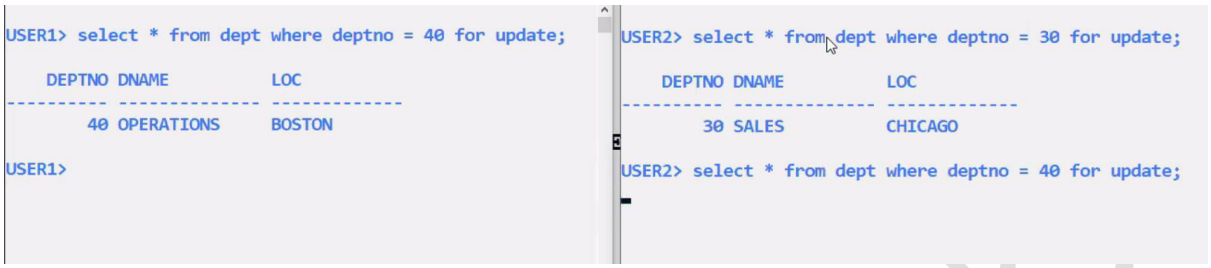
Select * from emp for update;

- Locks are automatically released when you rollback and commit

Lock the table

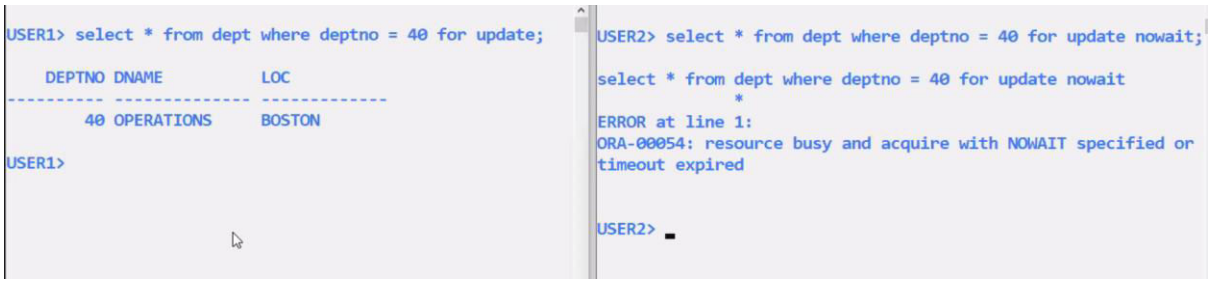
```
USER1> select * from emp for update;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7788 SCOTT      ANALYST         7566 09-DEC-82       3000                    20
      7876 ADAMS      CLERK           7788 12-JAN-83       1100                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10

14 rows selected.

USER1>
```

Select * from emp
Where deptno=10
For update; //by default it is wait

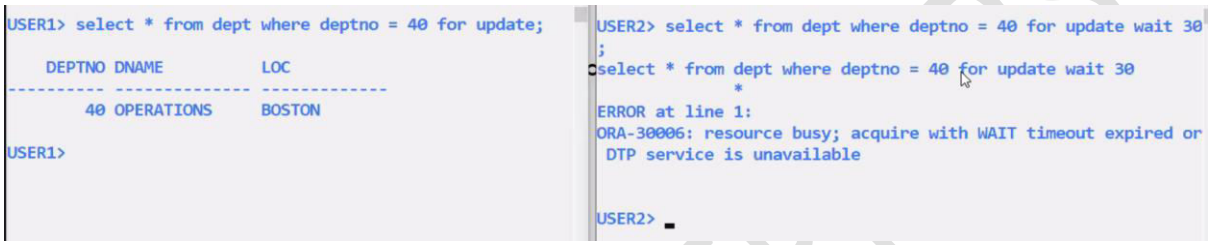


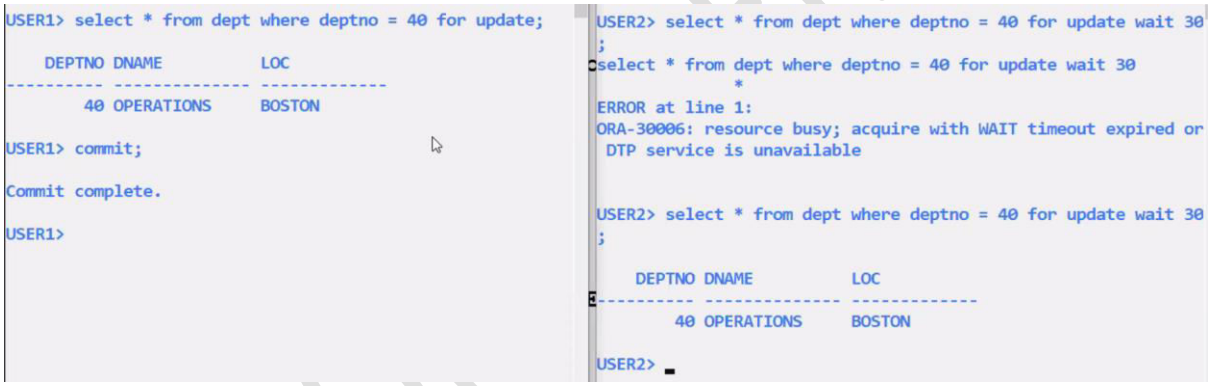
Select * from emp
Where deptno=10
For update wait;

Select * from emp
Where deptno=10
For update nowait;



Select * from emp
Where deptno=10
For update wait 60; //waiting time(Seconds)





MySQL workbench to try out row locking:-
Click on query (menu at the top)->new tab to current server ->click on it

Now you have 2 Query windows to try out row locking
If you get stuck in the request queue, to abort the operation:-
Click on Query(menu at the top)->click on stop

- Wait/nowait options are not available in MySQL
- In MySQL it is default Wait.

MySQL – SQL – Functions

Character functions

- These are applicable for char datatypes and varchar datatypes.

EMP Table

FNAME	LNAME
ARUN	PURUN
TARUN	ARUN
SIRAN	KIRAN
NUTAN	PURAN

```
Create table emp(
fname varchar(20)
lname varchar(20)
);
```

```
Select fname, lname from emp;
```

Concatenate function:

```
Select concat (fname, lname) from emp;
```

```
concat (fname, lname)
----------------------------
ArunPurun
TarunArun
Sirunkirun
NutanPuran
```

```
Select concat (fname, ‘ ’,lname) from emp;
concat (fname,’ ’,lname)
----------------------------
Arun Purun
```

Tarun Arun
Sirun kirun
Nutan Puran

In Oracle:-

Select concat(concat(fname,' '),lname) from emp;

- **Max upto 255** levels for **function within function** (common for all RDBMS)
- This upper of SQL can be exceeded with the help of Views

Select concat('Mr. ',fname,' ',lname) from emp;

Upper function:

Select concat(upper(fname), upper(lname)) from emp;

Select upper(concat(fname,lname)) from emp;

ARUNPURUN
TARUNARUN
SIRUNKIRUN
NUTANPURAN

Update emp set fname = upper(fname) ;

Solution for case insensitive query in Oracle
Select * from emp where upper(fname) = 'ARUN';

Lower function:

Select concat(lower(fname), lower (lname)) from emp;

Select lower (concat(fname,lname)) from emp;

arunpurun
tarunarun
sirunkirun
nutanpuran

Update emp set fname = lower (fname) ;

Solution for case insensitive query in Oracle
Select * from emp where lower (fname) = ‘ARUN’;

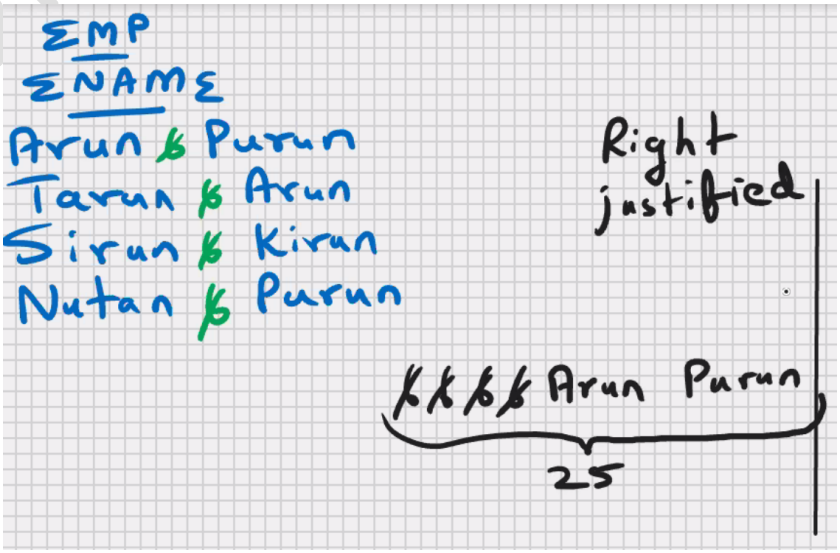
Initcap function: (not available in MySQL, available in Oracle)

Select initcap(fname) from emp;

Arun
Tarun
Sirun
Nutan

Lpad function:(Stands for left pad) Right justified

EMP table
Ename
Arun Purun
Tarun Arun
Sirun kirun
Nutan Puran



Select lpad(ename,25,’ ’) from emp;

```
SQL> select lpad(ename,25,' ') from emp;

LPAD(ENAME,25,'')
-------------------------------------------
                    KING
                   BLAKE
                   CLARK
                   JONES
                  MARTIN
                   ALLEN
                  TURNER
                   JAMES
                    WARD
                    FORD
                   SMITH
                   SCOTT
                   ADAMS
                  MILLER

14 rows selected.

SQL>
```

- Use for right justification

Select lpad(ename,25,’* ’) from emp;

```
SQL> select lpad(ename,25,'*') from emp;

LPAD(ENAME,25,'*')
-------------------------------------------
********************KING
********************BLAKE
********************CLARK
********************JONES
********************MARTIN
********************ALLEN
********************TURNER
********************JAMES
********************WARD
********************FORD
********************SMITH
********************SCOTT
********************ADAMS
********************MILLER

14 rows selected.
```

- Use for cheque printing

Rpad function:(Stands for right pad) Left justified

Uses

- Left justification of numeric data
- Convert varchar to char (convert variable length to fixed length)
- Cheque printing

```
SQL> select rpad(sal,25) from emp;

RPAD(SAL,25)
---------------------------
5000
2850
2450
2975
1250
1600
1500
950
1250
3000
800
3000
1100
1300
```

Homework : centre justified

Ltrim function:

Select ltrim(ename) from emp;

```
Arun Purun
   Tarun Arun
Sirun Kirun
 Nutan Purun
```

```
Arun Purun
Tarun Arun
Sirun Kirun
Nutan Purun
```

Uses

- Left justification

Rtrim function:

Select rtrim(ename) from emp;

- Remove the spaces from right side
- Right justification e.g. lpad(rtrim(ename)….)
- Convert char to varchar (convert fixed length to variable length)

Trim function:

Select trim(ename) from emp;

- Removes the blank spaces from both the sides

Substr function:

Select Substr(ename,3) from emp;
un Purun
run Arun
run kirun
tan Puran

- Used to extract a part of the string
- Substr('kharghar',5)->'ghar'

Select Substr(ename,3,2) from emp;
2 -> number of characters

un
ru
ru
ta

Select Substr(ename,-3) from emp;
run
run
run
run

Select Substr(ename,-3,2) from emp;
ru
ru
ru
ru

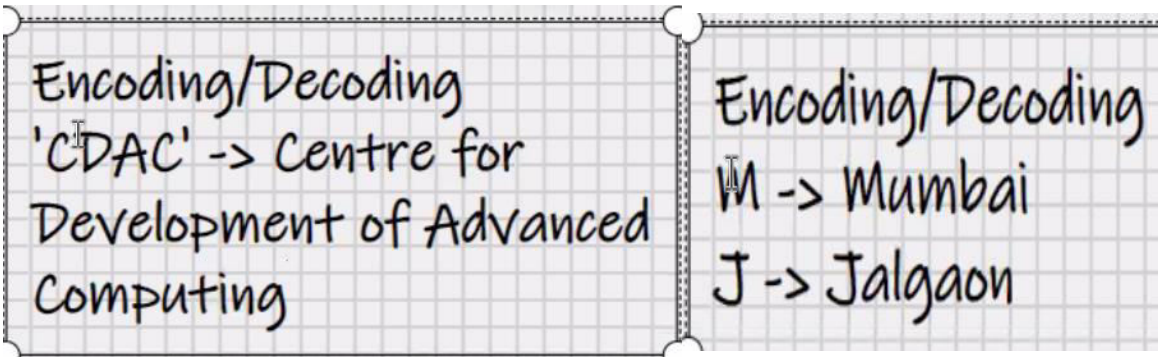
substr('New Mumbai',1,3) -> New
substr('New Mumbai',5) -> Mumbai

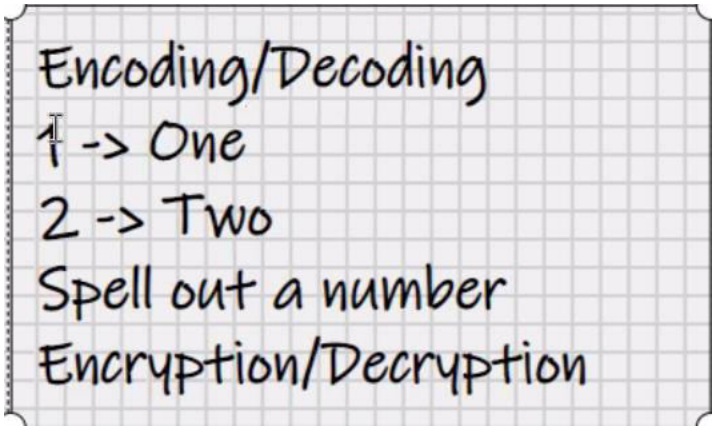
replace function:
select replace(ename,'un','xy') from emp;
un->xy

Arxy Purxy
Tarxy Arxy
Sirxy Kirxy
Nutan purxy

select replace(ename,'un') from emp; ->works in oracle (not supported by MySQL)

ar pur
tar ar
sir kir
nutan pur





Translate function:

Translate (ename,'un','xy') from emp;

U -> x

n -> y

arxy pxrxy

etc.

Translate (ename,'un','xyz') from emp;

U -> x

n -> y

->z

- Translate function is supported by oracle (not supported by MySQL)

Instr function:

Select instr(ename,'un') from emp;

Returns starting position of string

3

4

4

10

If string is not found returns 0.

uses

- Used to check if one string exists in another string

EMP table
Ename
Anirudha
Bhavesh
chaitanya
Dishi

Length function:
Select bhavesh (ename) from emp;
8
7
9
5

Ascii function:
Select ascii(ename) from emp;
65
66
67
68

Return ascii value of first character in string

Select ascii(substr(ename,2)) from emp;
Select ascii('z') from emp;

122
122
122
122

Select distinct ascii('z') from emp;

```
SQL> select ascii('z') from emp;

ASCII('Z')
----------
       122
       122
       122
       122
       122
       122
       122
       122
       122
       122
       122
       122
       122
       122

14 rows selected.
```

```
SQL> select distinct ascii('z') from emp;

ASCII('Z')
----------
       122

SQL>
```

Use mysql;
Select distinct ascii(‘z’) from user;

122

```
SQL> select ascii('z') from dual;

ASCII('Z')
----------
       122

SQL>
```

Select ascii(‘z’) from dual;

- Dual is a system table
- Dual is a dummy table
- It contains only 1 row and
- 1 column

Select substr(‘New Mumbai’,1,3) from dual;
Select 3*12 from dual;
Select ‘Welcome to CDAC Mumbai’ “MESSAGE” from dual;

Select char(65 using utf8) from dual;
A

- Where utf8 is the given character set for us English else default is binary character set

Soundex function:
Emp table
Ename
Arun
Bhavesh
Chaitanya

Select * from emp where soundex(ename)= soundex('Aruun');

A , e, I, o ,u ,y remove the vowels from the string and compares (work for simple names)

Number functions:
Emp table
Sal
1234.567
1848.019
1375.618
1752.156

In MySQL
Sal float

In Oracle
Sal number(7,3)

Round function:
Select round(sal) from emp;
1235
1848
1376
1752

Select round(sal,1) from emp;
1234.6
1848

Select round(sal,2) from emp;
1234.57
1848.02
Etc.

Select round(sal,-2) from emp;
1200
1800
1400
1800

Select round(sal,-3) from emp;
1000
2000
1000
2000

In MySQL

Truncate function:
Selelct truncate(sal,0) from emp;
Removes the decimal
1234
1848
1375
1752

Uses

- Date calculations and time calculations

Selelct truncate(sal,1) from emp;

1234.5
1848
1375.6
1752.1

Selelct truncate(sal,2) from emp;

1234.56
1848.01
1375.61
1752.15

Selelct truncate(sal,-2) from emp;

1200
1800
1300
1700

In oracle:-
Select trunc(sal) from emp;
Select trunc(sal,1) from emp;
Select trunc(sal,2) from emp;
Select trunc(sal,-2) from emp;

Ceil function :
Select ceil(sal) from emp;
If there is any value at all in the decimal then it will add 1 to the whole number

1235
1849
1376
1753
Uses

- Bill payments
- Interest calculation
- EMI payments
- Etc

Floor function:
Select floor (sal) from emp;
1234
1848
1375
1753

Select truncate(3.6,0), floor(3.6), truncate(-3.6,0), floor(-3.6) from dual; 3 3 -3 -4

Sign function:
Select sign(-15) from dual;
-1

If number is negative returns -1
If number is positive returns 1
If number is zero returns 0

Uses:

- Check if number is +ve or -ve
- Temperature
- Sensex

- Voltage
- Blood group
- Medical report
- Bank balance
- Profit and loss
- Distance

X and Y two unknown numbers
X=10
Y=20
Sign(x-y)
To find greater of two number use sign function

Lateral Thinking by Ayn Rand

Mod function:
Select mod(9,5) from dual;
4
Select mod(8.22,2.2) from dual;
1.62

Sqrt function:
Select sqrt(81) from dual;
9
Only works for positive number

Power function:
Select power(10,3) from dual;

1000
Select power(1000,1/3) from dual;
10

Abs function: (absolute number)

Select abs(-10)from dual;

Return positive value

10

Some more functions

Sin(x)

Cos(x)

Tan(x)

X is in radians

Ln(y)

Log(n,m) ->logn(m)

Date function

Emp table

Hiredate
2019-10-15
2019-12-31
2019-01-13

```
MySQL - SQL - Date Functions (MySQL ONLY)

a. 'YYYY-MM-DD'
b. 'YY-MM-DD' (00-69, 70-99)
c. 1st Jan 1000 AD to 31st 9999 AD
d. Date, Time, Datetime, Year
e. Date1-Date2
f. Datetime1-Datetime2
g. date is stored as a fixed-length number ->
number of days since 1st Jan 4712 BC
h. 7 Bytes
```

Select sysdate() from dual ; returns current date and time when the statement executed

- Sysdate() returns server date and time *"**abhi abhi**"*

Select now() from dual; returns current date and time When the statement began to execute

- now() returns server date and time

select sysdate(), now(), from dual;

select sysdate(), now(), sleep(10), sysdate() ,now() from dual;

```
1  select sysdate(), now(), sleep(10), sysdate(), now() from dual;
```

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

sysdate()	now()	sleep(10)	sysdate()	now()
2022-05-10 14:35:00	2022-05-10 14:35:00	0	2022-05-10 14:35:10	2022-05-10 14:35:00

Sysdate() -> date and time displays
Now()-> to maintain logs of operations of DML operations

Select adddate(sysdate(),1) from dual; show next day adding 24 hrs

```
3  select adddate(sysdate(),1) from dual;
```

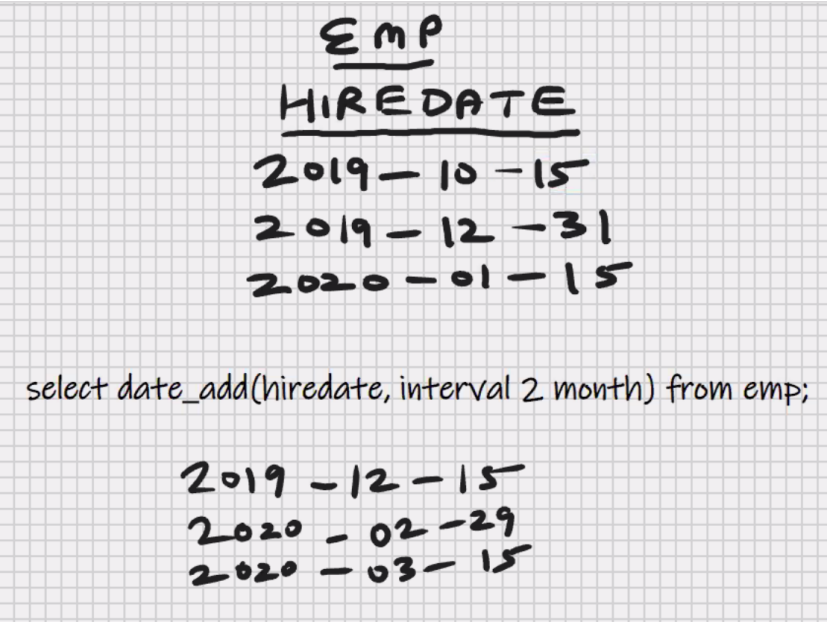
Result Grid | Filter Rows: | Export:

adddate(sysdate(),1)
2022-05-11 14:41:20

Select adddate(sysdate(),2) from dual; day after tomorrow
Select adddate(sysdate(),-1) from dual; yesterday
Select adddate(sysdate(),7) from dual; add week

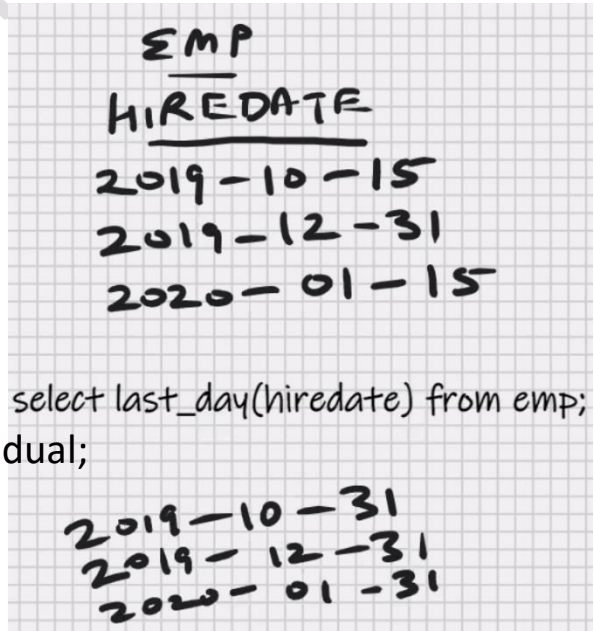
Select datedif(sysdate(),hiredate) from emp;
→ Returns number of dates between 2 dates;

Select date_add (hiredate, interval 2 month) from emp;



Select date_add (hiredate, interval -2 month) from emp;
Select date_add (hiredate, interval 1 year) from emp;
Select date_add (hiredate, interval -1 year) from emp;

Select last_day(hiredate) from emp;
This function is present in only
Oracle and mysql



Select dayname(sysdate()) from dual;
Tuesday

Select upper(dayname(sysdate())) from dual;
TUESDAY

Select addtime('2010-01-10' 11:00:00', '1') from dual;
Add 1 second to specified date and time

Select addtime('2010-01-10' 11:00:00', '1:00:00') from dual;
Add 1 hour to specified date and time

List functions(independent of datatypes)

Emp table

Ename	Sal	Comm
A	5000	500
B	6000	
C		700

```
SQL> select * from emp where comm = null;

no rows selected

SQL>
```

Select * from emp where comm=null;

- any comparisons done with null, returns null
- Pessimistict querying -> searching for null value

Select * from emp where comm is null;

```
SQL> select * from emp where comm is null;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7788 SCOTT      ANALYST         7566 09-DEC-82       3000                    20
      7876 ADAMS      CLERK           7788 12-JAN-83       1100                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10

10 rows selected.

SQL>
```

Select * from emp where comm!=null;

- any comparisons done with null, returns null

select * from emp where comm is not null;

```
SQL> select * from emp where comm != null;

no rows selected

SQL> select * from emp where comm is not null;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
```

- is null is a special operator

select sal+comm from emp;
5500
(Null)
(null)

- any operation done with null, returns null

in MySQL-
select sal+ ifnull(comm,0) from emp;
5500
6000
null

- if the comm is null returns 0;
- else return comm;
- end if;

select ifnull(sal,0) + ifnull(comm,0) from emp;
5500
6000
700

Ifnull(comm,0)
Ifnull(comm,100) -> if comm is null return 100
Ifnull(city,'Mumbai')
Ifnull(orderdate,'2022-04-01')

In oracle:-
Select nvl(sal,0) + nvl(comm, 0) from emp;

Nvl(comm,0)
Nvl(comm,100) -> if comm is null return 100
Nvl(city,'Mumbai')
Nvl(orderdate,'2022-04-01')

Emp table

Sal
1000
2000
3000
4000
5000

Greatest function:
Select greatest (sal,3000) from emp;
3000
3000
3000
4000
5000

Select greatest (sal,3000,4000,5000,6000,10000) from emp;
10000
10000

10000
10000
10000

Greatest (col1, col2, col3, …….., col255)
Greatest (val1, val2, val3, ……., val255)

Greatest (num1, num2, num3)
Greatest (‘str1’,’str2’,’str3’,’str4’)
Greatest (date1, date2, date3)

Set x = greatest(a,b,c,d);

Least function:
Select least(sal,3000) from emp;
1000
2000
3000
3000
3000

Least (col1, col2, col3, …….., col255)
Least (val1, val2, val3, ……., val255)

Least (num1, num2, num3)
Least (‘str1’,’str2’,’str3’,’str4’)
Least (date1, date2, date3)

Set x = least(a,b,c,d);

- greatest function used to set a lower limit on some value
- least function used to set a upper limit on some value

bonus= 10% sal, min bonus=300
select greatest(sal*01,300) “BONUS” from emp;
cashback = 10% amt, max cashback = 1000;
least(amt*0.1,1000) from orders;

CASE expression

EMP TABLE

SAL	DEPTNO
1000	10
2000	10
3000	20
4000	30
5000	40

Select
Case
When dept=10 then ‘Training’
When dept=20 then ‘Exports’
When dept=30 then ‘Sales’
Else ‘Others’
End “DEPTNAME”
From emp;

```
SQL> select deptno,
  2  case
  3  when deptno = 10 then 'Training'
  4  when deptno = 20 then 'Exports'
  5  when deptno = 30 then 'Sales'
  6  else 'Others'
  7  end "DEPTNAME"
  8  from emp;

    DEPTNO DEPTNAME
---------- --------
        10 Training
        30 Sales
        10 Training
        20 Exports
        30 Sales
        30 Sales
        30 Sales
        30 Sales
        30 Sales
        20 Exports
        20 Exports
        20 Exports
        20 Exports
        10 Training
```

Training
Training
Exports
Sales
Others

ELSE is optional in case expression
If else is not specified then for all other values,
It returns null values
It is like switch case statement

```
Select
Case
When dept = 10 then ‘Ten’
When dept = 20 then ‘Twenty’
When dept = 30 then ‘Thirty’
When dept = 40 then ‘Forty’
End  “DEPTNO”
From emp;
```

If deptno = 10 then HRA = 40% annual
If deptno= 20 then HRA = 30% annual
If deptno = 30 then HRA = 20% annual

```
Select deptno, ename, sal*12 annual,
Case
When deptno=10 then sal*12*0.4
When deptno=20 then sal*12*0.3
When deptno=30 then sal*12*0.2
Else sal*12*0.1
End “HRA”
From emp;
```

If sal>3000 then REMARK = ‘High Income’
If sal<3000 then REMARK = ‘Low Income’
If sal=3000 then REMARK = ‘Middle Income’

```
Select ename, sal
Case
When sign(sal-3000)=1 then ‘High Income’
When sign(sal-3000)=-1 then ‘Low Income’
Else ‘Middle Income’
End “REMARKS”
From emp
Order by 2;
```

MySQL – SQL – Environment functions

In MySQL-

Select user() from dual;

'dishi@localhost'

- used to maintain the logs of DML operations

emp

Ename	Sal	Col1	Col2	Col3
King	5000	Dishi	2022-05-10	Etc

Insert into emp values

('King',5000,user(),now(),etc);

Show character set;

In Oracle:

Select user from dual;

Group / Aggregate functions

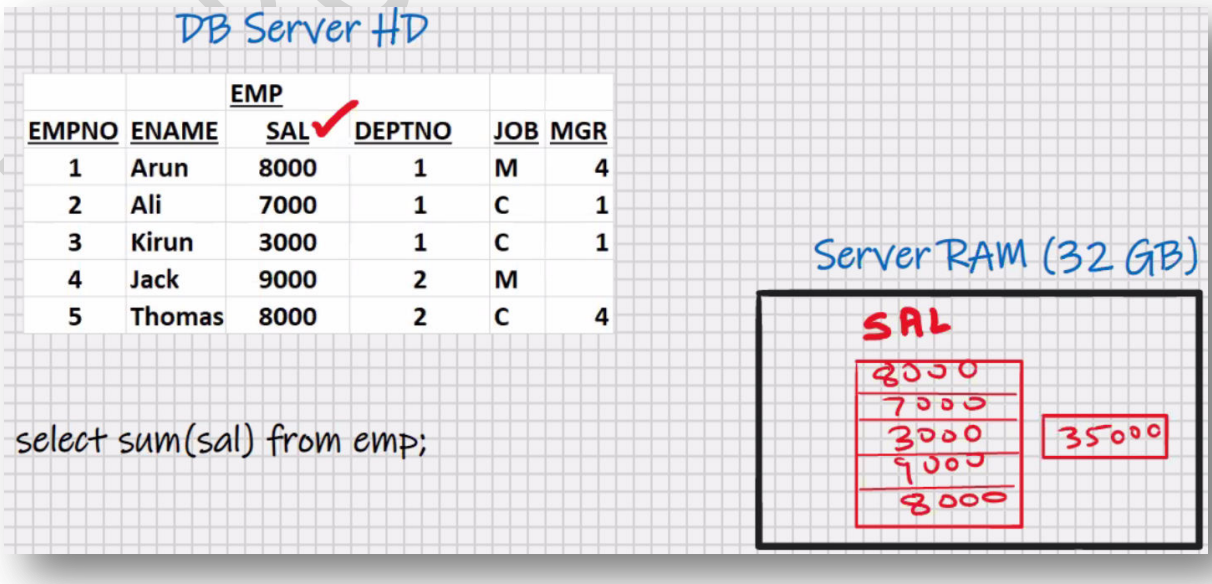
		EMP			
EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

Select
Case
When job = ‘M’ then ‘MANAGER’
When job = ‘C” then ‘CLERK’
End “JOB”
From emp;

Single row functions:
Operate on one row at a time:
Character, number, date, list, environment functions
e.g. upper(ename), round(sal), etc.

Multi-Row functions:
Operate on multiple rows at a time
e.g sum(sal), min(sal), max(sal), etc.

group functions:
sum function:
select sum(sal) from emp;
35000



Assumption 4th row sal is null:-

Sum function:

Select sum(sal) from emp; <- null values are not counted by group functions

26000

Select sum(ifnull(sal,0)) from emp; <- you may do this but not required

26000

Average function-avg:

Select avg(sal) from emp;

26000/4=6500

Select avg(ifnull(sal,0)) from emp;

26000/4=5200

Minimum function:

Select min(sal) from emp;

3000

Select min(ifnull(sal,0)) from emp;

0

Maximum function:

Select max(sal) from emp;

8000

Select max(ifnull(sal,0)) from emp;

0

Count function:

Select count (sal) from emp;

- Return number of rows where sal is not having null value

4

Select count (*) from emp;

- Returns count of total number of rows in the table

5

Select count(*)-count(sal) from emp;

1

Select max(sal)/min(sal) from emp;

8000/3000=2.67

Select sum(sal)/count(*) from emp;

26000/5=5200

Select avg(ifnull(sal,0)) from emp;

26000/5=5200

Assumption 4th row sal Is 9000:-

Select sum(sal) from emp

Where deptno=1;

18000

- Where clause is used for searching
- Searching takes place in DB server HD

Select avg(sal) from emp
Where job = ‘C’;
6000

COUNT-QUERY:
Counting the number of Query hits:-

Select count(*) from emp ->**VERY IMPORTANT**
Where sal>7000;
3

Sum(column)
avg(column)
min(column)
max(column)
count(column)
count(*)
stddev(column)
variance(column)

min(sal), min(ename), min(hiredate)
max(sal), max(ename), max(hiredate)
count(sal), count(ename), count(hiredate)

- used for all datatypes

select sum(sal), avg(sal), min(sal), max(sal) from emp;

- multiple group functions can be selected

```
SQL> select count(*), min(sal), max(sal), sum(sal), avg(sal) from emp;

  COUNT(*)   MIN(SAL)   MAX(SAL)   SUM(SAL)   AVG(SAL)
---------- ---------- ---------- ---------- ----------
        14        800       5000      29025 2073.21429
```

- summary of my table

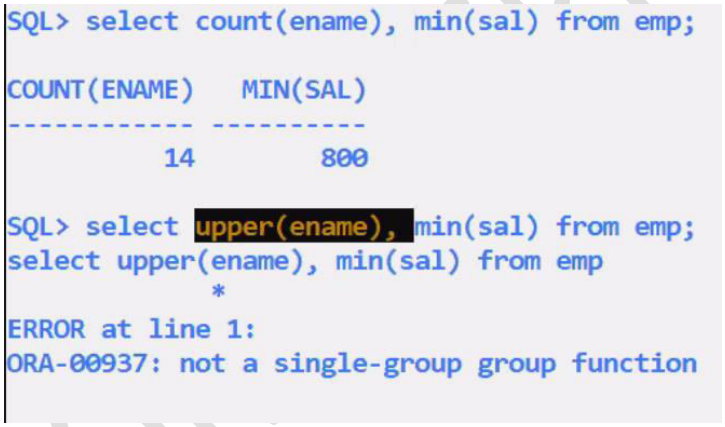
3 RESTRICTIONS:-

Select ename, min(sal) from emp; <- error in oracle (works in MySQL but the output is meaningless)

1. you cannot select a column of table by itself, alongwith a group function

---

select upper(ename), min(sal) from emp; <- error in oracle (works in MySQL but the output is meaningless)



2.you cannot select a single row function alongwith a group function

---

Select * from emp

Where sal>avg(sal); -> Error in MySQL and Oracle

3.you cannot use group function in the where clause (Where clause is used for searching; the searching takes place in DB server HD at the time of searching in the table the avg(sal) doesn't exist; the avg (sal) is calculated by MySQL afterwards (after the rows are brought into server ram))

My-SQL- SQL-group by clause(Very Important)

- used for grouping

select sum(sal) from emp;

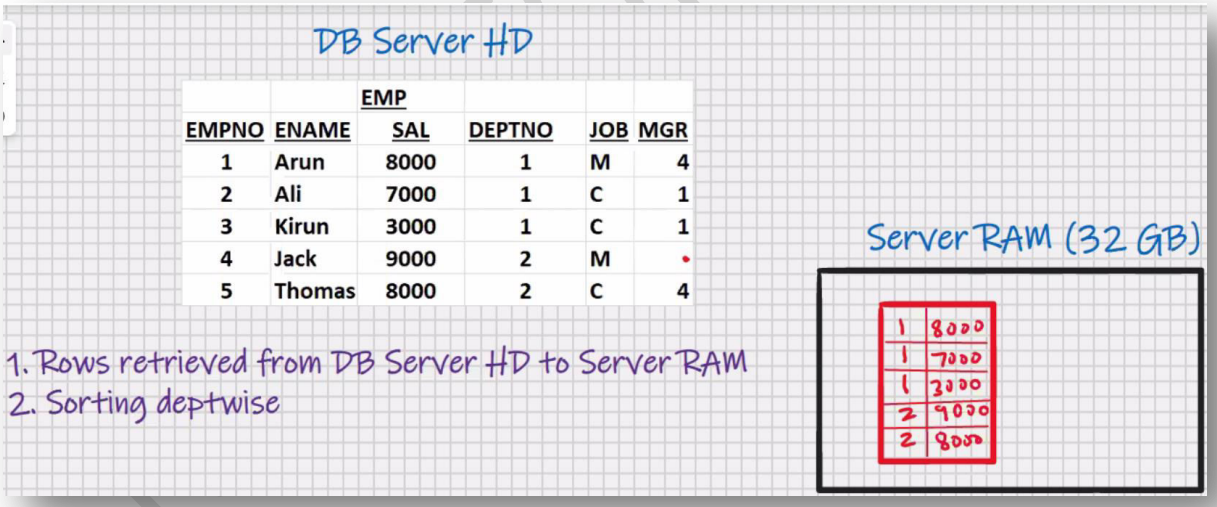
35000

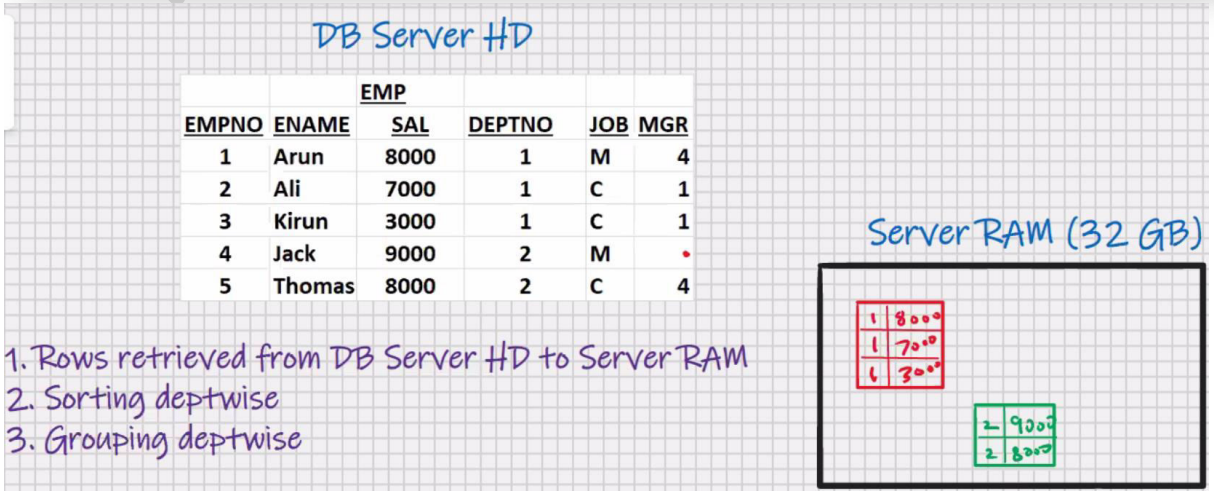
Select sum(sal) from emp
Where deptno=1;

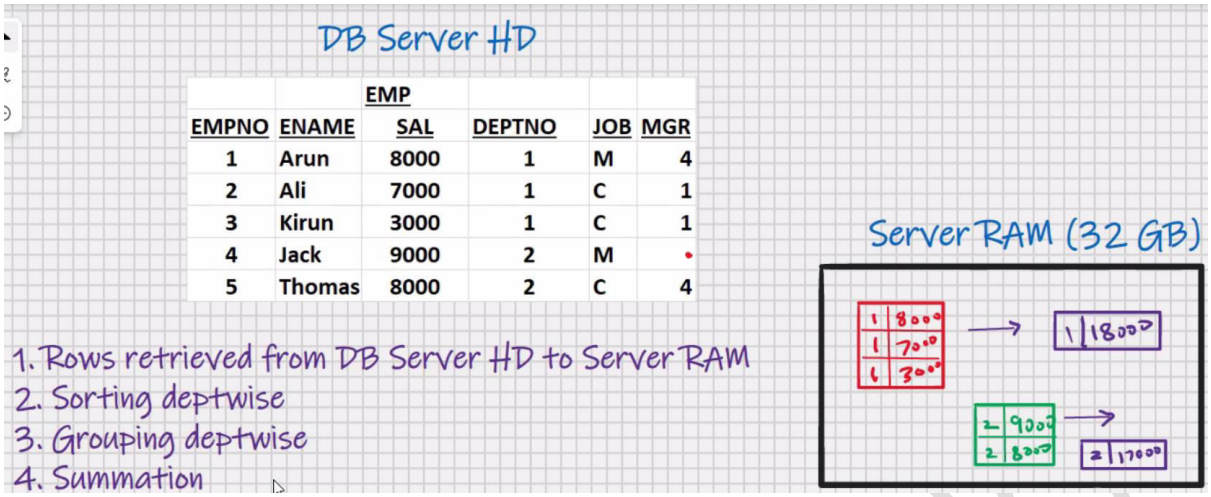
18000

Select deptno, sum(sal) from emp
Group by deptno;

DEPTNO	SUM(SAL)
1	18000
2	17000







1. Besides the group function, whichever column is present in select clause; it has to be present in group by clause

Select deptno, sum(sal) from emp; -> ; <- error in oracle (works in MySQL but the output is meaningless)

Select deptno, sum(sal) from emp
Group by deptno;

2. whichever column is present in group by clause it may or may not be present in select clause

Select sum(sal) from emp
Group by deptno;
SUM(SAL)

18000
8000

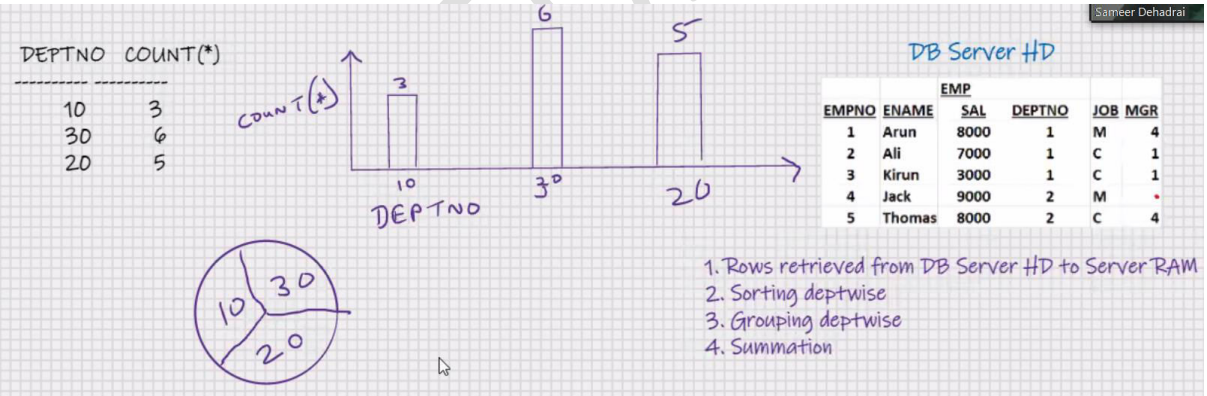
Select deptno, sum(sal) from emp
Group by deptno;

Select deptno, max(sal) from emp
Group by deptno;

Select deptno,min(sal) from emp
Group by deptno;

Select deptno, count(*) from emp
Group by deptno;

2D query-> any select statement with a group by clause is known as a 2D query, because you can plot a graph from the output e.g. x-y grapgh, bar graph, pie chart, etc.
#1 Oracle Graphics



Select job, sum(sal) from emp
Group by job;

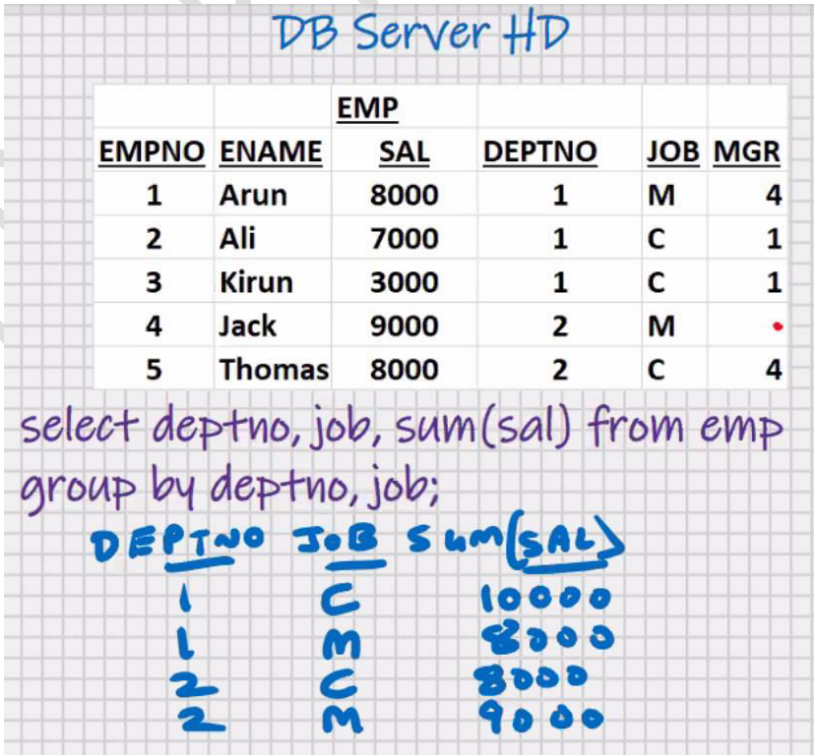
Select job, sum(sal) from emp
Where sal>7000
Group by job;

- **WHERE clause has to be specified before the group by clause**

- WHERE clause is used for searching ; the searching takes place in DB server HD
- WHERE clause is used to restrict the rows
- WHERE clause is used to retrieve the rows from DB server HD to server RAM

```
select deptno, sum(sal) from emp
where sal > 7000
group by deptno;

DEPTNO   SUM(SAL)
------   --------
1         8000
2        17000

select deptno, sum(sal) from emp
where job = 'C'
group by deptno;
```

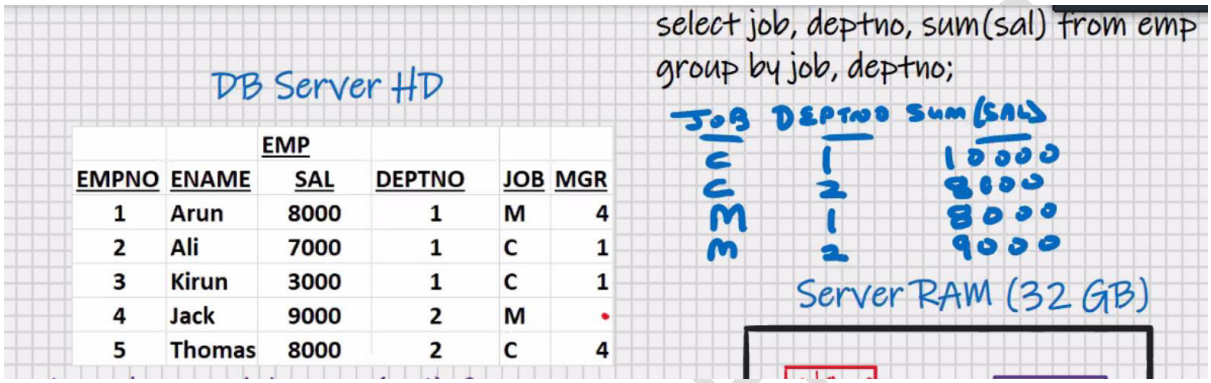
Select deptno, job, sum(sal) from emp
Group by deptno, job;

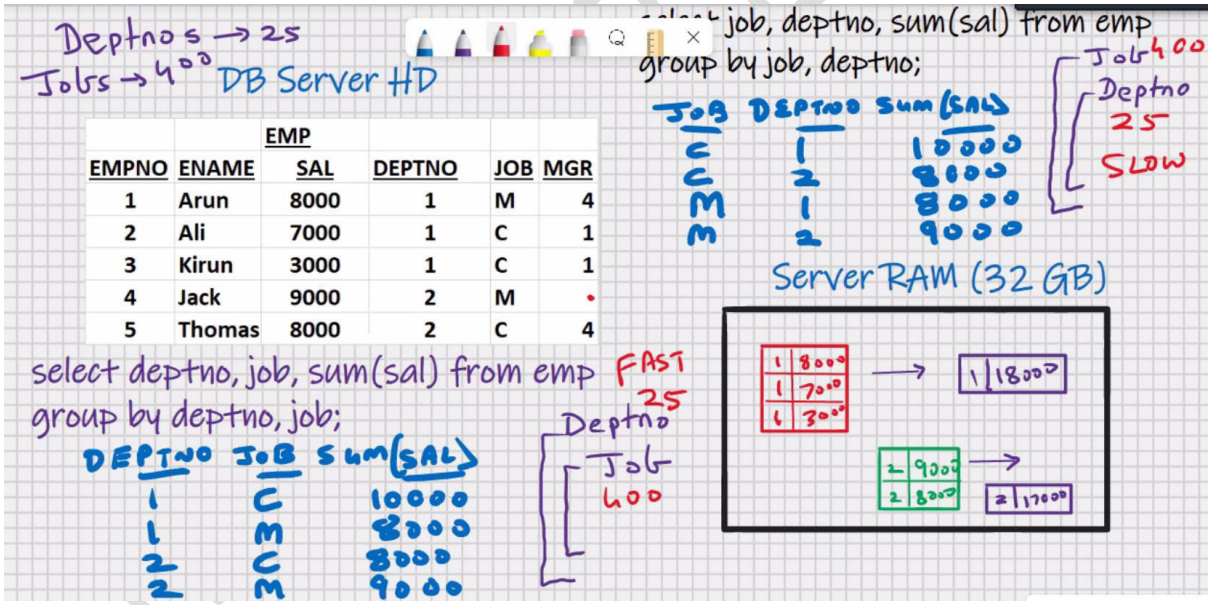


- No upper limit on the number of columns in Group by clause

Select …….
Group by country, state, city;

Select job, deptno sum(sal) from emp
Group by job, deptno;





Select job, deptno sum(sal) from emp
Group by deptno, job;

- The position of columns in select clause and the position of columns in group by clause need not be the same

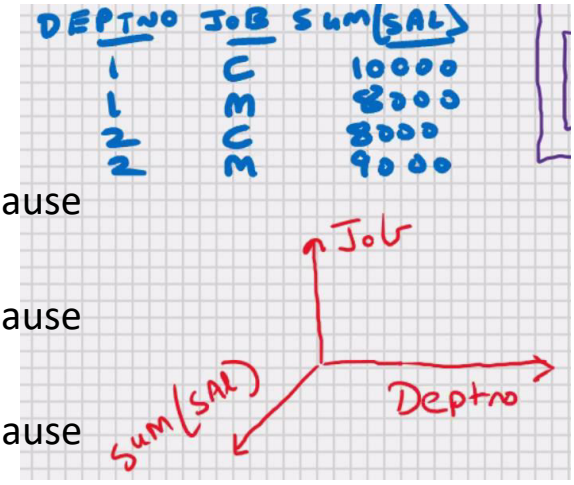
- The position of columns in select clause will determine the position of columns in the output(this you will write as per user requirements)
- The position of columns in group by clause will determine sorting order, the grouping order, the summation order, and hence the speed of processing

```
select sum(sal), job, deptno from emp
group by deptno, job;

*       the position of columns in SELECT clause and the position of columns
        in GROUP BY clause need not be the same
*       the position of columns in SELECT clause will determine the position
        of columns in the output (this you will write as per User
        requirements)
*       the position of columns in GROUP BY clause will determine sorting
        order, the grouping order, the summation order, and hence the speed of
        processing
```

```
select .....................................
group by city, state, country, district;          <- SLOW

select .....................................
group by country, state, district, city;          <- FAST
```

Countries are less in number compare to states, districts, cities

Select deptno, job, sum(sal) from emp
Group by deptno, job;



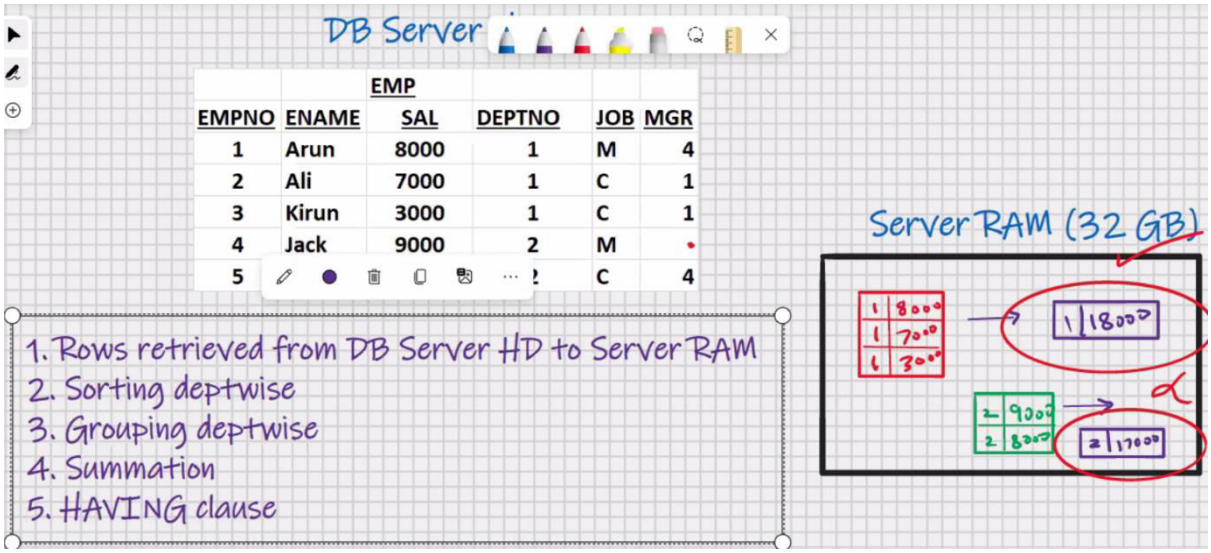
- If you have 1 column in group by clause
  It is 2D Query
- If you have 2 column in group by clause
  It is 3D Query
- If you have 3 column in group by clause
  It is 4D Query

- Known as **Multidimensional queries**(also known as **spatial queries**)

```
select deptno, sum(sal) from emp
group by deptno;
```

DEPTNO	SUM(SAL)
1	18000
2	17000

```
select deptno, sum(sal) from emp
group by deptno;
having sum(sal) > 17000
```



DEPTNO	SUM(SAL)
1	18000

- Having clause works after the summation is done
- Having clause is step number 5

```
select deptno, sum(sal) from emp
group by deptno;
having sum  > 17000                    -> Error
```

```
select deptno, sum(sal) from emp
where sal>7000
group by deptno;
```

- Where clause is used for searching
- Searching takes place in DB server HD
- Where clause is used to restrict the rows
- Where clause is used to retrieve the rows from DB server HD to server Ram
- Having clause works after the summation is done

```
select deptno, sum(sal) from emp
group by deptno
having where deptno = 1;
```
-> this will work but it is inefficient

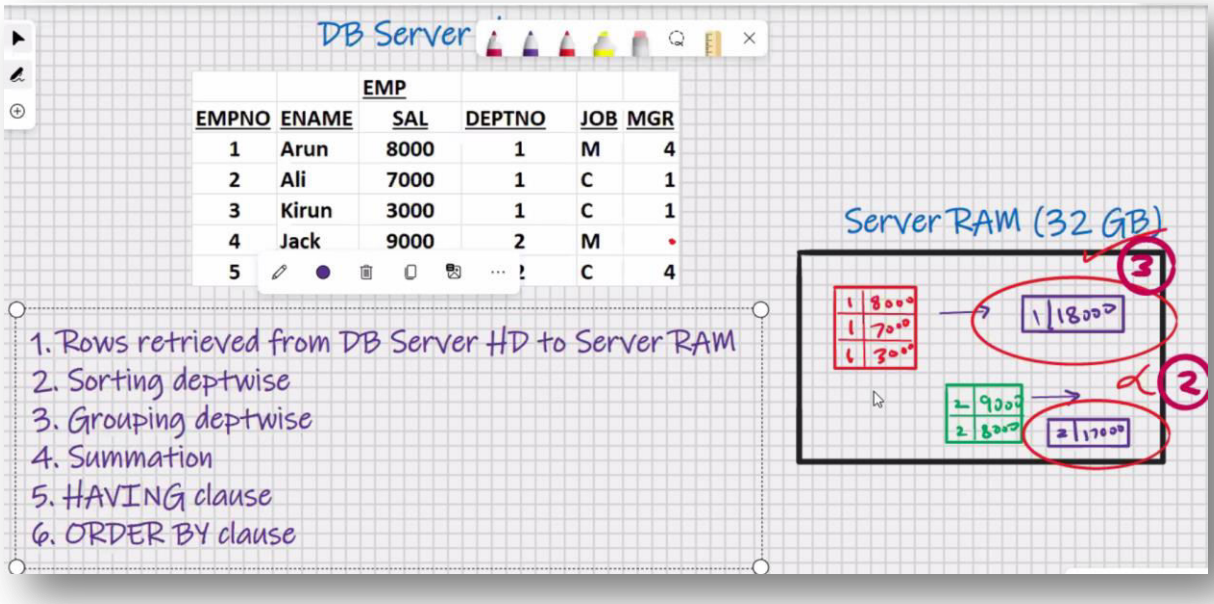
- Whichever column is present in select clause it can be used in **having** clause
- Its recommended that only **group functions** should be used in **having** clause

```
select deptno, sum(sal) from emp
group by deptno
having sum(sal)>17000 and sum(sal)<25000;
OR
select deptno, sum(sal) from emp
group by deptno
having sum(sal) between 17001 and 24999;

select deptno, sum(sal) from emp
group by deptno
having count(*)=3;
```

```
select deptno, sum(sal) from emp
group by deptno
order by sum(sal);

DEPTNO     SUM(SAL)
----------     ----------
    2              18000
    1              17000
```

```
select deptno, sum(sal) from emp
group by deptno
order by 2;
DEPTNO     SUM(SAL)
----------     ----------
    2              18000
    1              17000
```

- Order by is the last clause in select statement



```
Select deptno, sum(sal) from emp
Where sal>7000
Group by deptno
Having sum(sal) >10000
Order by 1;
DEPTNO    SUM(SAL)
----------    ------------
    2          17000
```

```
Select deptno, count(*), min(sal), max(sal), sum(sal) from emp
Group by deptno
Having count(*) >=3
Order by 1;                    ->  where 1 is column number
```

In Oracle:-

```
Select deptno, sum(sal) from emp
Group by deptno;

DEPTNO    SUM(SAL)
----------    ------------
    1             18000
    2             17000
```

```
Select sum(sal) from emp
Group by deptno;

SUM(SAL)
------------
18000
17000
```

Select max(sum(sal)) from emp
Group by deptno; -> **nesting of group functions is allowed only in oracle RDBMS**

```
MAX(SUM(SAL))
-------------
18000
```

In MySQL:

Select deptno, sum(sal) from emp
Group by deptno;

```
DEPTNO    SUM(SAL)
-----------    -------------
    1              18000
    2              17000
```

select max(sum_sal) from
(select sum(sal) sum_sal from emp
Group by deptno) as tempp;

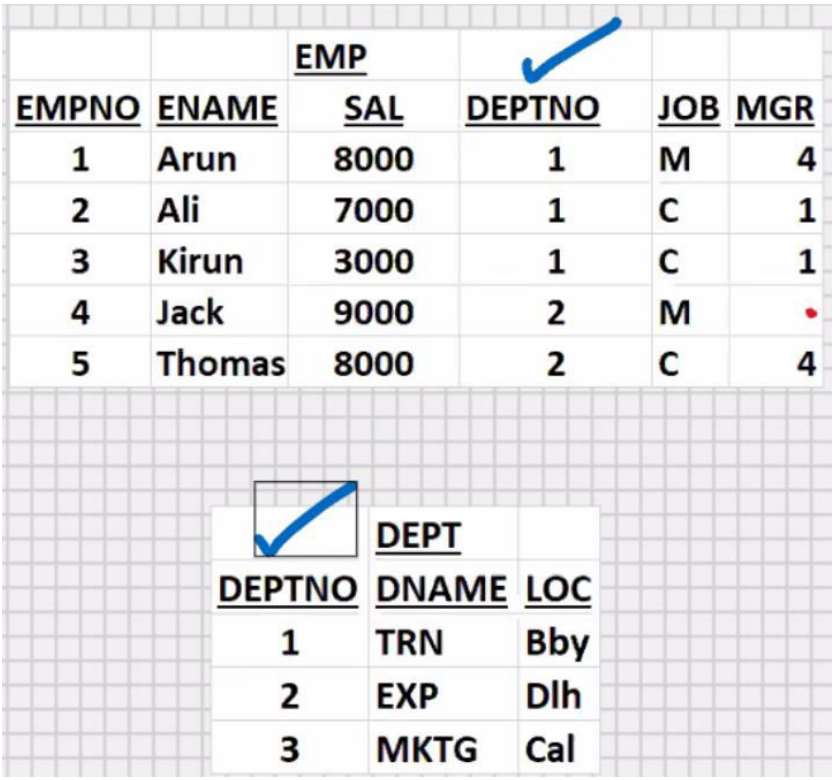
```
Max(sum_sal)
-------------------
18000
```

Matrix Report:

Select deptno, count(*), min(sal), max(sal),sum(sal) from emp
Group by deptno
Order by 1;

MySQL – SQL – JOINS

- Very important topic for interview and job
- Data redundancy-> unnecessary duplication of data(wastage of HD space)
- Normalisation



EMP

EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

DEPT

DEPTNO	DNAME	LOC
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

Select ename, deptno from emp;

```
+--------+--------+
| ename  | deptno |
+--------+--------+
| Arun   |      1 |
| Ali    |      1 |
| Kiran  |      1 |
| Jack   |      2 |
| Thomas |      2 |
+--------+--------+
```

Dept -> driving table(must contain less no. of rows (work fast))

Emp -> driven table

<<-----

select ename, dname from emp, dept
where dept.deptno = emp.deptno;
(tablename.columnname)

ename	dname
---	---
Arun	Training
Ali	Training
Kiran	Training
Jack	Exports
Thomas	Exports

- To view the columns of 2 or more tables

  select ename, dname from emp, dept
  where emp.deptno = dept.deptno ;

- In order for the join to work faster, preferably the driving table should be table with “Lesser” number of rows
- Full table scan – entire table will be searched

select ename, dname from emp, dept
where emp.deptno = dept.deptno
order by dname;

select dname,ename from emp,dept
where dept.x=emp.y;

- The common column in both the Tables is DEPTNO; the common column In both the tables, the columnname need

EMP

EMPNO	ENAME	SAL	[redacted]	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

DEPT

[redacted]	DNAME	LOC
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

Not be the same in both the tables, because the same column may have a different meaning elsewhere(e.g. import export)

- What matter is that the datatype of the common column in both the tables should be the same

Select dname, loc, ename, job, sal from emp, dept
Where dept.deptno=emp.deptno
Order by 1;

Select * from emp, dept
Where dept.deptno=emp.deptno
Order by 1;

- Deptno column will display twice

Select dname, loc, ename, job, sal, **deptno** from emp, dept
Where dept.deptno=emp.deptno
Order by 1; →> error for deptno column ambiguity

Select dname, loc, ename, job, sal, dept.**deptno** from emp, dept
Where dept.deptno=emp.deptno
Order by 1;

Select dname, loc, ename, job, sal, emp.**deptno** from emp, dept
Where dept.deptno=emp.deptno
Order by 1;

Select dept.dname, emp.loc, emp.ename, emp.job, emp.sal, emp.**deptno** from emp, dept
Where dept.deptno=emp.deptno
Order by 1;

- Always use tablename.columnname in select statement

Select dept.dname, emp.loc, emp.ename, emp.job, emp.sal, emp.**deptno** from emp, dept
Where x = y // different name then not required to use tablename
Order by 1;

Select deptno, sum(sal) from emp
Group by deptno;

DEPTNO	SUM(SAL)
1	18000
2	17000

Select dname, sum(sal) from emp, dept
Where dept.deptno= emp.deptno
Group by dname;

DEPTNO	SUM(SAL)
TRN	18000
EXP	17000

Select upper(dname), sum(sal) from emp, dept
Where dept.deptno= emp.deptno
Group by upper(dname)
Having sum(sal)>10000
Order by 2 desc;

DEPTNO	SUM(SAL)
TRN	18000
EXP	17000

Types of Joins (5)

1. **Equijoin**

- Join based on equality condition
- Shows matching rows of both the tables
- Uses:
- Data is stored in multiple tables and you want to view the columns of multiple tables
- View dname, ename
- View cname, sname
- Dname, order_details
- Etc

```
Select dname, ename from emp, dept
Where dept.deptno = emp.deptno;
```

2. **Inequijoin**

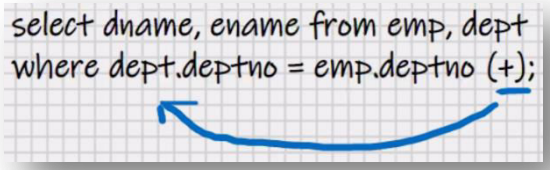
- Join based on inequality condition

```
Select dname, ename from emp, dept
Where dept.deptno != emp.deptno;
```

ename	dname
Arun	Exports
Ali	Exports
Kiran	Exports
Arun	Marketing
Ali	Marketing
Kiran	Marketing
Jack	Marketing
Thomas	Marketing
Jack	Training
Thomas	Training

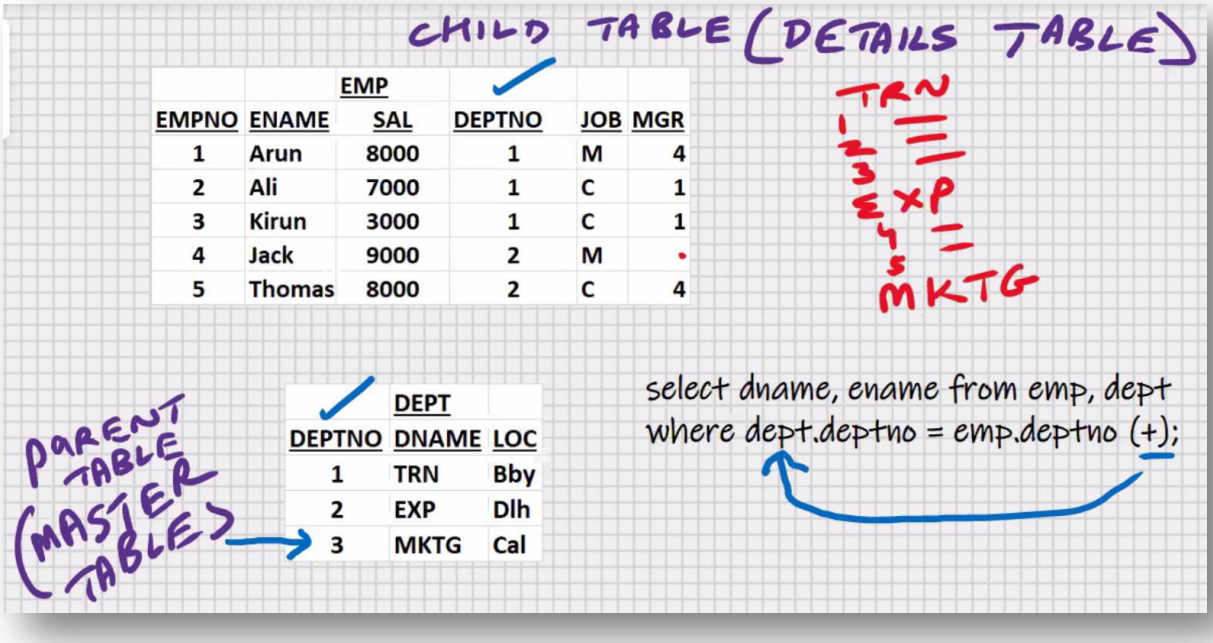
- Shows non-matching rows of both the tables
- Uses:
- **EXCEPTION REPORTS**

3. **Outerjoin(works only in oracle)**
   (dept-> do while emp-> for)
   Select dname, ename from emp, dept
   Where dept.deptno = emp.deptno **(+)**;
   LHS RHS



DNAME	ENAME
TRN	ARUN
TRN	ALI
TRN	KIRUN
EXP	JACK
EXP	THOMAS
MKTG	.

- Shows matching rows of both the tables plus non matching rows of “Outer” table
- OUTER table -> table which is on Outer side of (+) sign
- OUTER table -> table which is on opposite side of (+) sign



- Uses:
- **Master-detail report (Parent – child Report)**

Select dname, ename from emp, dept dept-> for
Where dept.deptno **(+)** = emp.deptno; emp->do while
LHS RHS

- **Half Outerjoin**:You can + sign on any side(LHS or RHS) but not on both the sides.
  - **Right outerjoin (+sign on RHS of where clause)**
  - **Left outerjoin (+sign on LHS of where clause)**
- **Full Outerjoin(based on Nested do while loop)**

Select dname, ename from emp, dept
Where dept.deptno **(+)** = emp.deptno
union
Select dname, ename from emp, dept
Where dept.deptno = emp.deptno (+);

DNAME	ENAME
TRN	Arun
TRN	Ali
TRN	Kirun
EXP	Jack
EXP	Thomas
MKTG	.
.	SCOTT

  - Shows matching rows of both the tables plus Non matching rows of both the tables

ANSI syntax for full Outerjoin:

- Supported by all RDBMS including Oracle except for MySQL

Select dname, ename from emp **full** outer join dept
On (dept.deptno = emp.deptno);

To achieve full outerjoin in MySQL :

Select dname, ename from emp **right** outer join dept
On (dept.deptno = emp.deptno)
union
Select dname, ename from emp **left** outer join dept
On (dept.deptno = emp.deptno);

ANSI syntax for Right Outerjoin:

- Supported by all RDBMS including MySQL and Oracle

Select dname, ename from emp **right** outer join dept
On (dept.deptno = emp.deptno);

ANSI syntax for Left Outerjoin:

- Supported by all RDBMS including MySQL and Oracle

Select dname, ename from emp **left** outer join dept
On (dept.deptno = emp.deptno);

- (+) sign for outerjoin only works in Oracle RDBMS
- (+) sign for outerjoin Is not supported by any other RDBMS

INNER join -> by default every join is an Inner join
Using the (+) or by using the keyword “Outer”, it becomes an Outer join

- Do not mention inner join in interviews, unless explicitly asked by interviewer

4. **Cartesian join (also known as Cross join)**

- Join without a where clause
- Every row of driving table is combined with each and every row of driven table
- Similar to taking cross product of two tables
- Shows all the combinations
- Uses:
- Printing purposes
- E.g in students table you have all the students names, in subjects table you have all the subjects name; when you're printing the marksheets, every student name is combined with each and every subject name

Dept -> driving table
Emp -> driven table
Select dname, ename from emp , dept; -> **fast(I/O) between DB server HD and Server RAM is 3(the lesser the I/O faster it will be)**

Select dname, ename from dept, emp ; **-> slow(I/O) between DB server HD and Server RAM is 5(the more the I/O slower it will be)**

ename	dname
Arun	Training
Arun	Exports
Arun	Marketing
Ali	Training
Ali	Exports
Ali	Marketing
Kiran	Training
Kiran	Exports
Kiran	Marketing

Jack	Training
Jack	Exports
Jack	Marketing
Thomas	Training
Thomas	Exports
Thomas	Marketing

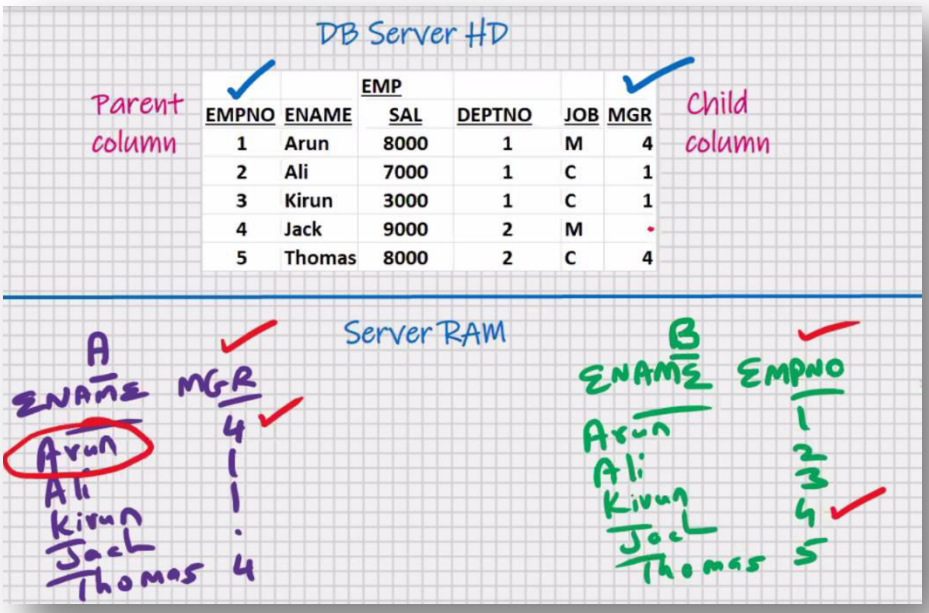
5. **Self join**
    - Joining a table to itself
    - Used when parent column and child column both are present in the same table

a -> driving table b -> driven table

Select a.ename, b.ename from emp b, emp a

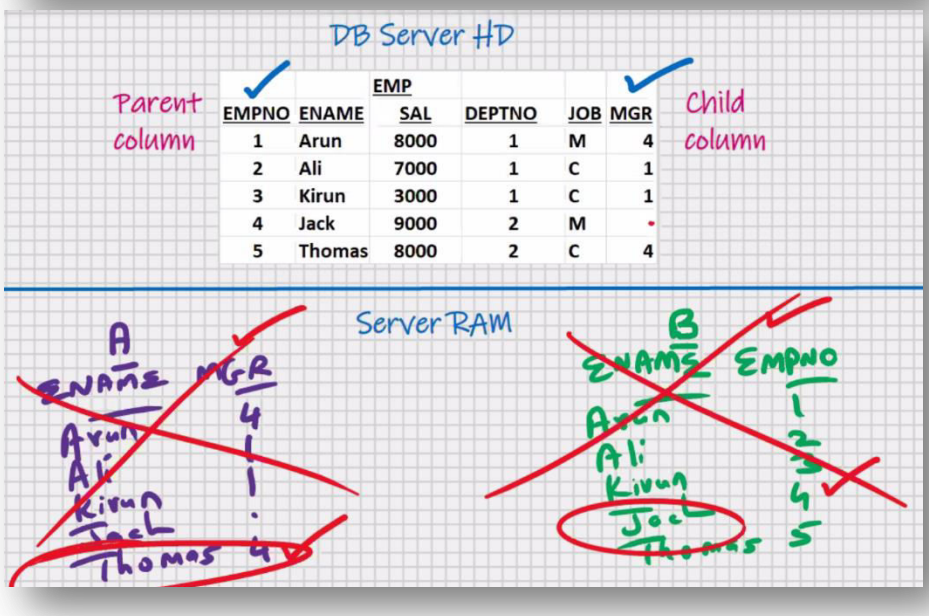
Where a.mgr = b.empno;

Employee	Manager
Arun	Jack
Ali	Arun
Kiran	Arun
Thomas	Jack



Temporary tables

Destroy after completion



- Uses:
- Employee name , manager names
- **Slowest join (based on Recursion)**

Select dname, ename from emp e, dept d
Where d.deptno = e.deptno; -> BAD Example

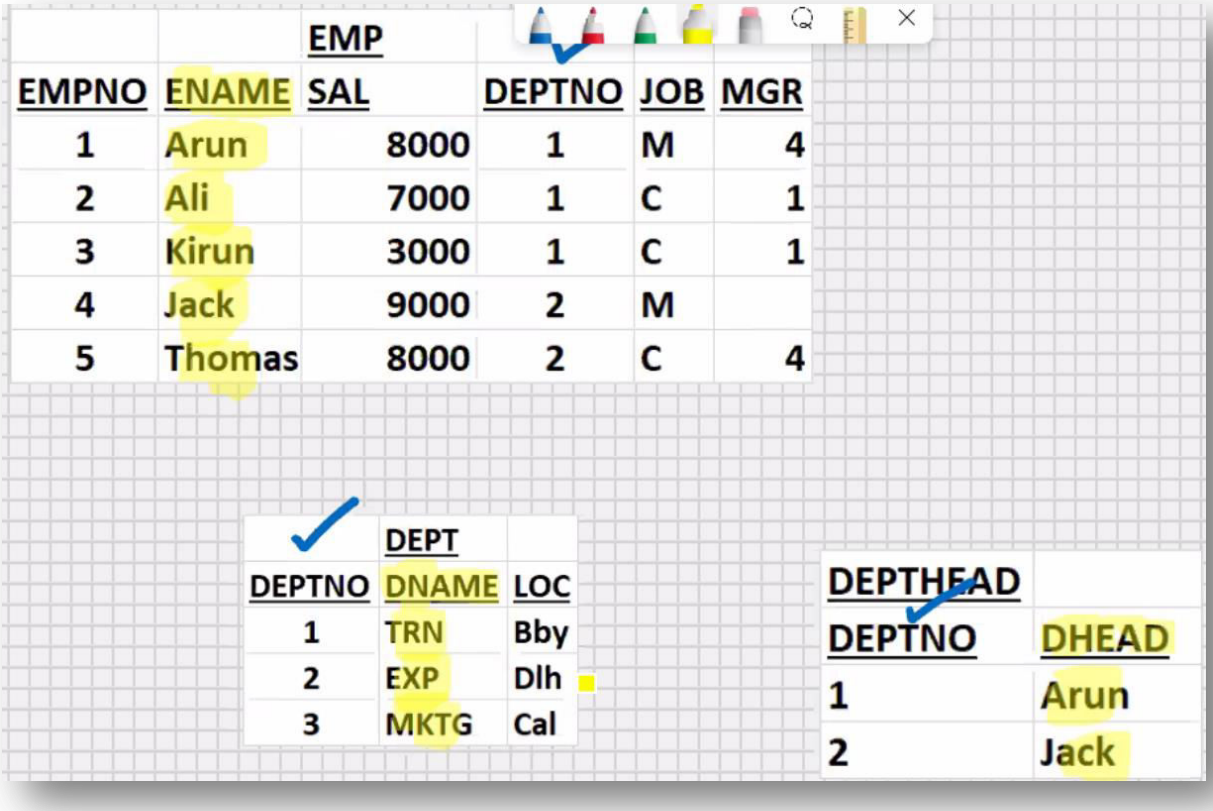
- **Do not give alias to tablename unnecessarily, do so in rare event you write a self-join**

1.Equijoin

- Most frequently used (>90%) ; hence it is also known as Natural join

2.Inequijoin (Non-equijoin)
3.Outerjoin
4.Cartesian Join (also known as Cross join) (fastest join because there is no where clause and hence there is no searching involved)
5.Self join (Based on Recursion) (slowest join)

- **Internally all joins are based on Nested for loop**
  Except for the Outerjoin where one or more of the loops may be do while loop(rest are all for loop)



EMP

EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

DEPT

DEPTNO	DNAME	LOC
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

DEPTHEAD

DEPTNO	DHEAD
1	Arun
2	Jack

<<----- <<-----

Select dname, ename, dhead from emp, dept, depthead
Where depthead.deptno=dept.deptno
And dept.deptno=emp.deptno;

dname	ename	dhead
Training	Arun	Arun
Training	Ali	Arun
Training	Kiran	Arun
Exports	Jack	Jack
Exports	Thomas	Jack

Types of Relationships

1	: 1	(dept : depthead) or (depthead : dept)
1	: many	(dept : emp) and (Depthead : emp)
Many	: 1	(emp : dept) and (emp : depthead)
Many	: Many	(projects : emp) or (emp : projects)

EMP

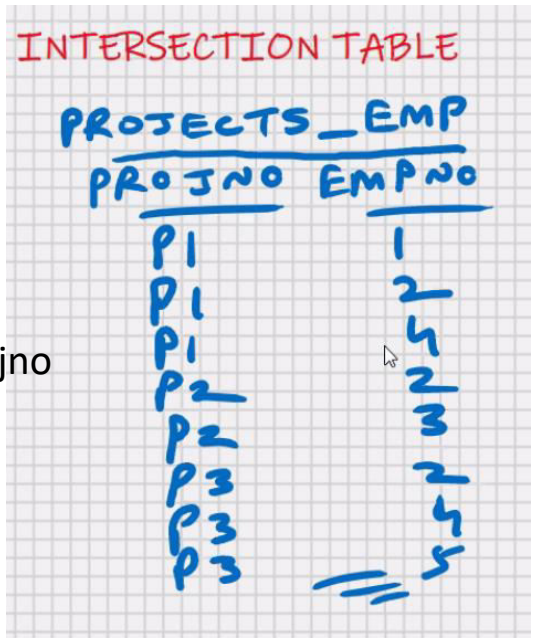
EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

PROJECTS

PROJNO	CLIENTNAME	DESCRIP
P1	Deloitte	CGS
P2	BNP Paribas	Macros S/w
P3	Morgan Stanley	AMS
P4	ICICI Bank	PPS
P5	AMFI	Website Dev

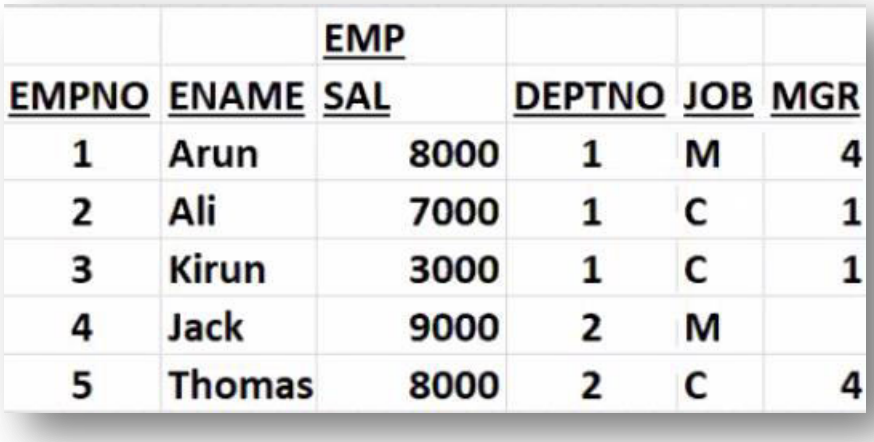
Intersection table

- Intersection table is required
  For Many : Many Relationship




```
Select clientname, ename,descript
From projects_emp, emp , projects
Where projects.projno = projects_emp.projno
And emp.empno = projects_emp.empno
Order by 1, 2, 3;
```

MySQL – SQL – Sub-queries



		EMP			
EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

- Query within query (select within select) (Nested Queries)

  Display the ename who is receiving sal = min(sal):

  ```
  Select ename from emp       //main query (outer query) (parent)
  Where sal =
   (select min(sal) from emp) ;  //sub-query(inner query) (child)
  ```

  ```
  ENAME
  ----------
  Kirun
  ```

  ```
  Select ename from emp       //main query (outer query) (parent)
  Where sal =
   (select min(sal) from emp
  Where deptno =
  (select deptno from …….
  Where job =
  (select ……)
  )
  ) ;
  ```

- Max upto 255 levels for subqueries (this limit of SQL can be exceeded with the help of views)

```
Select ename from emp
Where sal =
 (select min(sal) from emp
Where deptno = 1) ;
```

- Join is faster than sub query(when you solve a problem using join, you solve the problem using one select statement; when you solve a problem using subquery, you require two or more select statements)
- The more the number of select statements, the slower it will be

Display the 2nd largest SAL:

```
Select max(sal) from emp
Where sal <
(select max (sal) from emp);
```

EMP

EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

Display all the rows who belong to the same deptno as ‘Thomas’:

```
Select * from emp
Where deptno =
(select deptno from emp
Where ename = ‘Thomas’);
```

Display all the rows who are doing the same job as ‘Kirun’:

```
Select * from emp
Where job =
(select job from emp
Where ename = ‘Kirun’);
```

Using sub-queries with DML commands

```
Delete from emp
Where deptno =
(select deptno from emp
Where ename = 'Thomas');

Update emp set sal = 10000
Where job =
(select job from emp
Where ename = 'Kirun');
```

- Above 2 commands will work in oracle
- Above 2 commands are not supported by MySQL

Using sub-queries with DML commands:-

- In MySQL you cannot update or delete from a table from which you are currently selecting

Solution for MySQL:

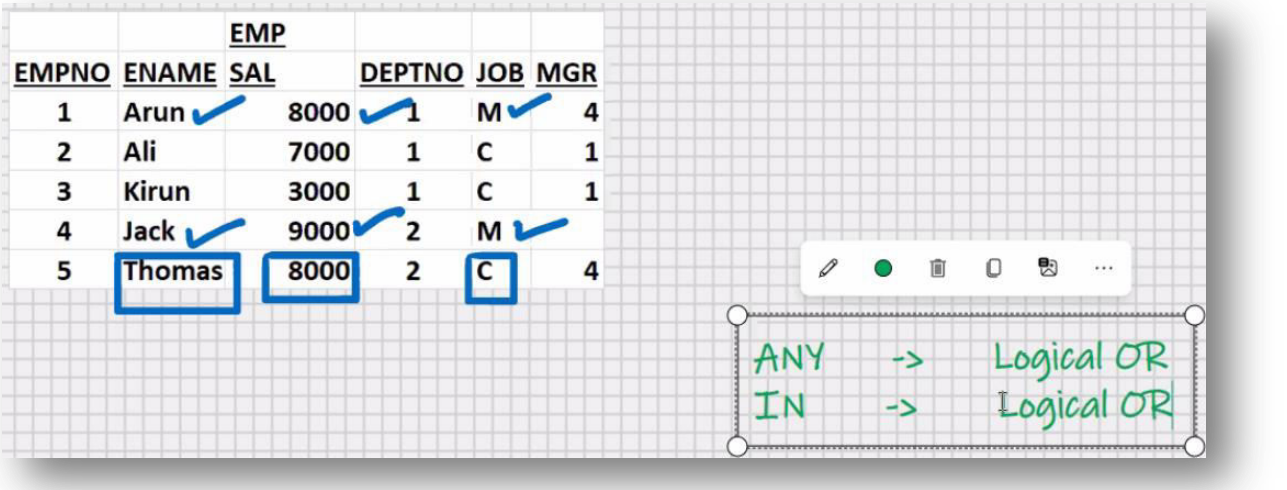
```
Delete from emp
Where deptno=
(select tempp.deptno from
(select deptno from emp
Where ename = 'Thomas')as tempp);

Update emp set sal =10000
Where job =
(select temp.job from emp
(select job from emp
Where ename = 'Kirun')as temp);
```

Multi-row Sub-queries (Sub-query returns multiple rows)

```
Select * from emp
Where sal = any          (8000,9000)
(select sal from emp
Where job = 'M');
```

```
Select * from emp
Where sal  in            (8000,9000)
(select sal from emp
Where job = 'M');
```

- Any operator is overloaded that's why in operator is faster
- But any operator is powerful



EMP

EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4


```
Select * from emp
Where sal  >=            (8000)
(select min(sal) from emp
Where job = 'M');
```

To make it work faster:-

1. Join is faster than sub-query

Try to solve the problem using join

2. Tyr to reduce the number of levels of sub-queries
3. Try to reduce the number of rows returned by sub-query

***Assumption 3rd row sal is 13000**

```
Select * from emp
Where sal  > all          (8000,9000)
(select sal from emp
Where job = 'M');
```

EMP

EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

All is a special operator performs **logical And**

```
Select * from emp
Where sal  >              (9000)
(select max(sal) from emp
Where job = 'M');
```

*** Assumption 3rd row sal is 3000**

Using sub-query in the having clause:-
In Oracle : -

Display the dname that is having Max(sum(sal))

EMP

EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

DEPT

DEPTNO	DNAME	LOC
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

```
Select deptno, sum(sal) from emp
Group by deptno;

Deptno      sum(sal)
----------  -----------
1           18000
2           17000

Select sum(sal) from emp
Group by deptno;

Sum(sal)
------------
18000
17000

Select max(sum(sal)) from emp
Group by deptno;

Max(sum(sal))
-------------------
18000

Select deptno, sum(sal) from emp
Group by deptno
Having sum(sal) =
(select max(sum(sal)) from emp
Group by deptno);

Deptno      sum(sal)
----------  -----------
    1           18000
```

```
Select dname, sum(sal) from emp, dept
Where dept.deptno = emp.deptno
Group by dname
Having sum(sal) =
(select max(sum(sal)) from emp
Group by deptno);
```

Dname	sum(sal)
TRN	18000

In MySQL:-

```
Select sum(sal) from emp
Group by deptno;
```

```
Sum(sal)
-----------
18000
17000
```

```
Select max(sum_sal) from emp
(select sum(sal) sum_sal from emp
Group by deptno ) as tempp;
```

```
Max(sam_sal)
-----------------
18000
```

```
Select deptno, sum(sal) from emp
Group by deptno
Having sum(sal) =
(select max(sum_sal) from
(select sum(sal) sum_sal from emp
Group by deptno ) as tempp);
```

Deptno	sum(sal)
1	18000

```
Select dname, sum(sal) from emp, dept
Where dept.deptno = emp.deptno
Group by dname
Having sum(sal) =
(select max(sum_sal) from
(select sum(sal) sum_sal from emp
Group by deptno) as tempp);
```

Dname	sum(sal)
TRN	18000

MySQL – SQL – Correlated Sub-query (Very Important)

- Using the exists operator (**EXISTS** is a special operator)

EMP

EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

DEPT

DEPTNO	DNAME	LOC
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

Display the dnames that contains employees:-
Display the dnames that do not contains employees:-

Solution 1:-

Select deptno from emp;
1
1
1
2
2

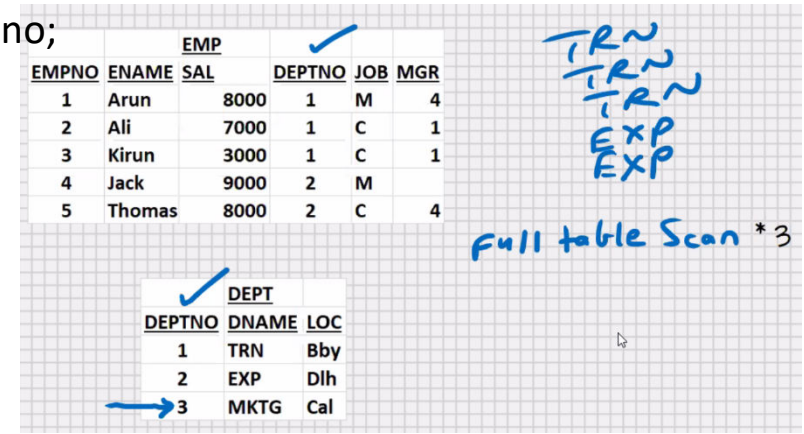
Select distinct deptno from emp;
1
2
Select distinct deptno from dept
Where deptno = any (1,2)
(Select distinct deptno from emp);

TRN
EXP
Select distinct deptno from dept
Where deptno in (1,2)
(Select distinct deptno from emp);
TRN
EXP

Select distinct deptno from dept
Where deptno not in (1,2)
(Select distinct deptno from emp);
MKTG

Solution 2:-

Select dname from emp, dept
Where dept.deptno = emp.deptno;



TRN
TRN
TRN
EXP
EXP

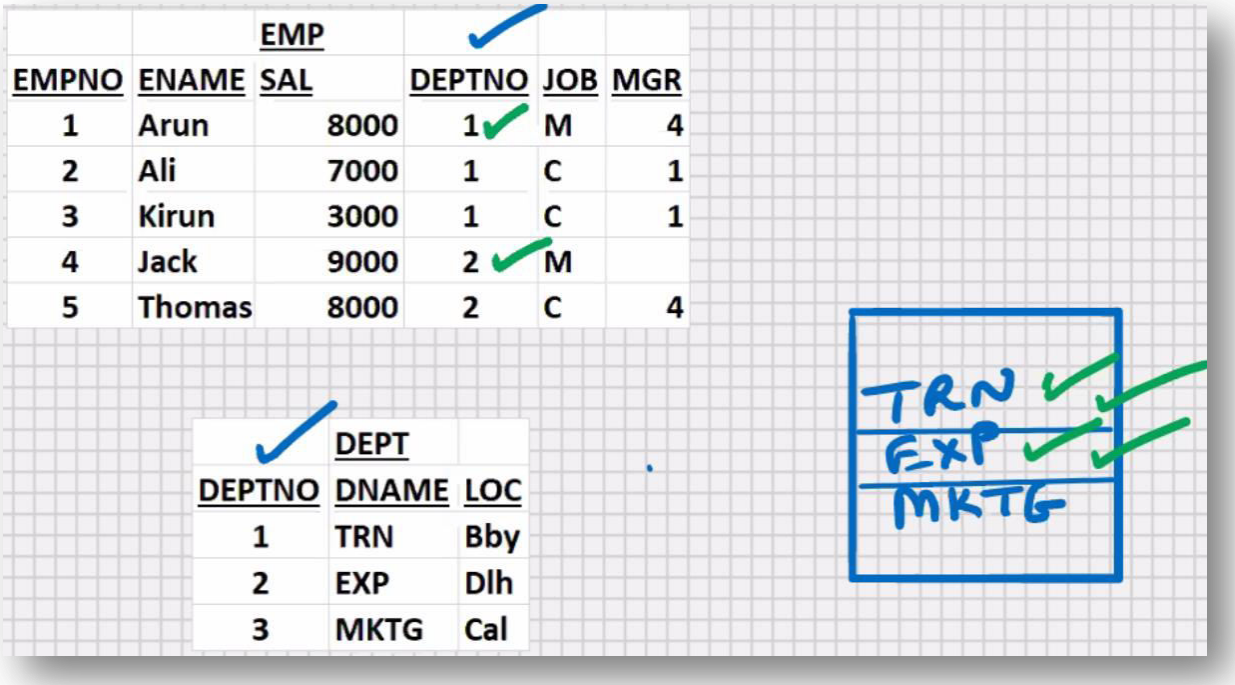
Select distinct dname from emp, dept
Where dept.deptno = emp.deptno;

TRN
EXP

Solution 3:-

- If you have a join along with subquery to make it work faster, use correlated subquery(use the EXISTS operator)
- This is the exception when sub query is faster than join

Select dname from dept where **exists**
(select deptno from emp
Where dept.deptno = emp.deptno);
TRN
EXP



EMP

EMPNO	ENAME	SAL	DEPTNO	JOB	MGR
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	3000	1	C	1
4	Jack	9000	2	M	
5	Thomas	8000	2	C	4

DEPT

DEPTNO	DNAME	LOC
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

- First the main query is executed
- For every row returned by main query, it will run the sub-query once the sub-query returns a boolean TRUE or FALSE value
- if subquery returns TRUE value, then main query is executed for that row
- if subquery returns FALSE value, the main query is not executed for that row

- unlike earlier we don't use **DISTINCT** here, hence no sorting takes place in server RAM; this speeds it up
- unlike earlier, the number of full table scans is reduced; this further speeds it up

Select dname from dept where **not exists**
(select deptno from emp
Where dept.deptno = emp.deptno);
MKTG

MySQL – SQL – Set Operators

- based on set theory
- foundation of RDBMS is based on set theory(founder of RDBMS (1968)-> Dr. Codd)
- data is intersection of row and column

EMP1

EMPNO	ENAME
1	A
2	B
3	C

EMP2

EMPNO	ENAME
1	A
2	B
4	D
5	E

```
Select empno, ename from emp1
        Union
Select empno, ename from emp2;
```

EMPNO	ENAME
1	A
2	B
3	C
4	D
5	E

Union-> **will combine the output of both the select statements, and it will suppress the duplicates**

- upper limit is 255 for joining select statement in union

Select **empno1**, ename from emp1
Union
Select empno2, ename from emp2;

EMPNO1	ENAME
1	A
2	B
3	C
4	D
5	E

Select empno1, ename from 1 where……..
Union
Select empno2, ename from 2 group by …….;(structure must match)

Select empno1, ename from 1
Union
Select empno2, ename from 2

Order by 1; // used order by clause after second select

- Order by clause is applicable to the final output

EMPNO1	ENAME
1	A
2	B
3	C
4	D
5	E

Union all -> **will combine the output of both the select statements , and the duplicates are not suppressed**

Select empno, ename from 1

Union all

Select empno, ename from 2;

Order by 1;

Empno	ename
1	A
1	A
2	B
2	B
3	C
4	D
5	E

```
Select empno, ename from 1
        intersect
Select empno, ename from 2;
```

Empno	ename
1	A
2	B

Intersect :- **will return what is common in both the select statements, and the duplicates are suppressed**

```
Select empno1, ename from 1
        minus
Select empno2, ename from 2;
```

Empno1	ename
3	C

minus :- will return what is present in the select statements, and not present in the select statement, and the duplicates are suppressed

```
select job from emp where deptno = 10
        minus
select job from emp where deptno = 20;
JOB
-----
PRESIDENT
```

```
select job from emp where deptno = 10
        intersect
select job from emp where deptno = 20;
JOB
-----
MANAGER
CLERK
```

Union, union all, intersect , minus

```
Select …….
        Union
Select …….
        Union all
Select …….
        intersect
Select …….
        minus
Select …….
        Union
Select …….
        Order by 1;
```

- Max upto 255 select statements
- (this limit of SQL can be exceeded with the help of views)

Union, union all -> available in all RDBMS(including MySQL)
Intersect, minus -> available in Oracle , **not available in MySQL**

- Multiple select statements; brackets for changing the precedence, **not supported in MySQL**

Pseudocolumns

- Pseudocolumns are fake columns (virtual columns)

e.g

a. computed columns (ANNUAL = sal*12)

b. Expressions (NET_EARNINGS = sal+comm)

c. Functions-based columns (e.g. TOTAL = sum(sal))

RDBMS supplied Pseudocolumns

Select ename, sal from emp;

Row indentifier

Select **rowid**, ename, sal from emp;

- Rowid is the row address
- Rowid is the address of the row in the DB server HD
- Rowid is the actual physical memory location where that row is stored in the DB Server HD
- Rowid is fixed length encrypted string of 18 characters
- When you select from a table , the order of rows in the ouptput will always be in the ascending order of rowid.
- When you update a row if the row length is increasing if the free space is not available at the current address, then the rowid may change
- No two rows of any table in the DB can have the same rowid
- Rowid works as an unique identifier for every row in the DB server HD

```
select rowid, ename, sal from emp;

ROWID                  ENAME                SAL
------------------ ---------- ----------
AAAS1HAABAAAdzxAAA KING                5000
AAAS1HAABAAAdzxAAB BLAKE               2850
AAAS1HAABAAAdzxAAC CLARK               2450
AAAS1HAABAAAdzxAAD JONES               2975
AAAS1HAABAAAdzxAAE MARTIN              1250
AAAS1HAABAAAdzxAAF ALLEN               1600
AAAS1HAABAAAdzxAAG TURNER              1500
AAAS1HAABAAAdzxAAH JAMES                950
AAAS1HAABAAAdzxAAI WARD                1250
AAAS1HAABAAAdzxAAJ FORD                3000
```

```
SQL> select rowid, ename, sal from emp where rowid = 'AAAS1HAABAAAdzxAAA';

ROWID              ENAME              SAL
------------------ ---------- ----------
AAAS1HAABAAAdzxAAA KING             5000

SQL>
```

- **You can use rowid to update or delete duplicate rows in the table**

Uses: Rowid is used internally by MySQL and Oracle:-

1. To Distinguish between 2 rows in the DB server HD
2. Rowid works as an unique identifier for every row in the DB server HD
3. For Row locking
4. To manage the indexes
5. To manage the cursors
6. Row management etc.

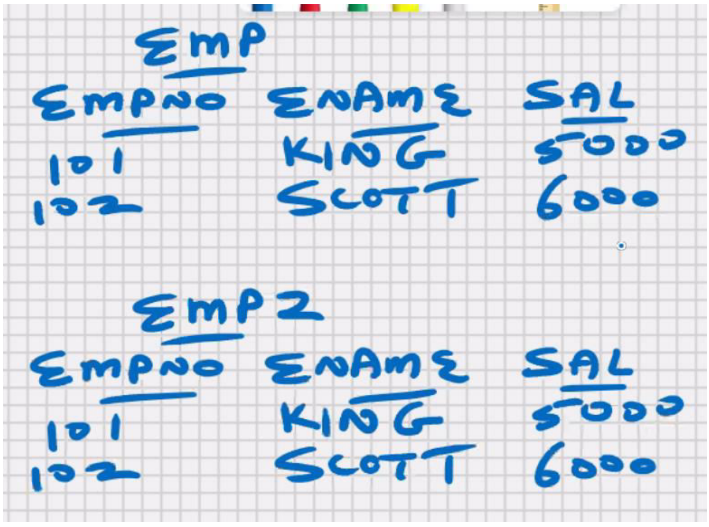
- Rowid is present in oracle and you can view it
- **Rowid is present in MySQL but you cannot view it**

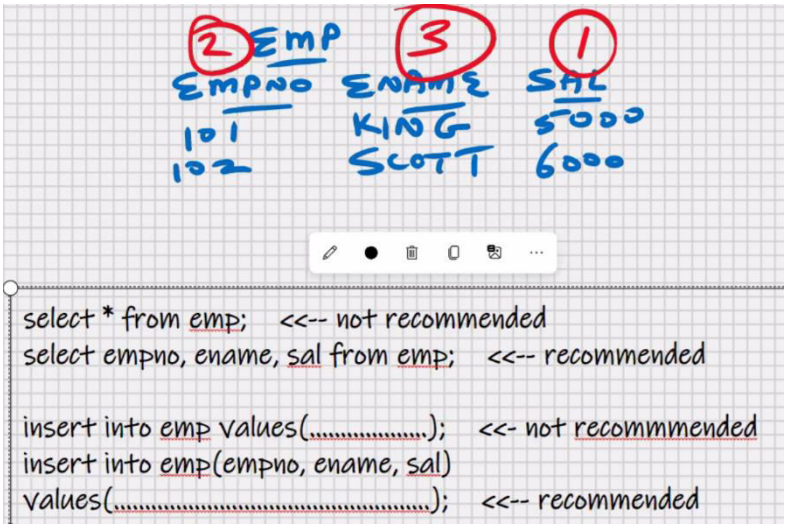
MySQL – SQL – ALTER TABLE (DDL COMMAND)

- Rename a table
- Add a column to the table
- Drop a column
- Increase width of column

Indirectly :

- Reduce the width of column
- Change the datatypes of column
- Copy the rows from one table into another table
- Copy a table (for testing purposes)
- Copy only the structure of table
- Rename a column
- Change the position of columns in table structure (because of null values) (for storage considerations)





- Rename a table

In oracle:

Rename emp to employees;

- Rename is a DDL command (extra in MySQl and oracle)

In MySQL:-

Rename table emp to employees;

- Add a column to the table

Alter table emp add gst float;

Rename table emp to employees;

- Drop a column

Alter table emp drop column gst;

- Increase width of column

Ename varchar (20) -> ename varchar (30)

Alter table emp modify ename varchar(30);

Indirectly :

- Reduce the width of column

In MySQL:

Alter table emp modify ename varchar (20);

- Data will get truncated (*that's dangerous yaar*)

In Oracle:

Alter table emp add x varchar (25);

Update emp set x = ename, ename = null;

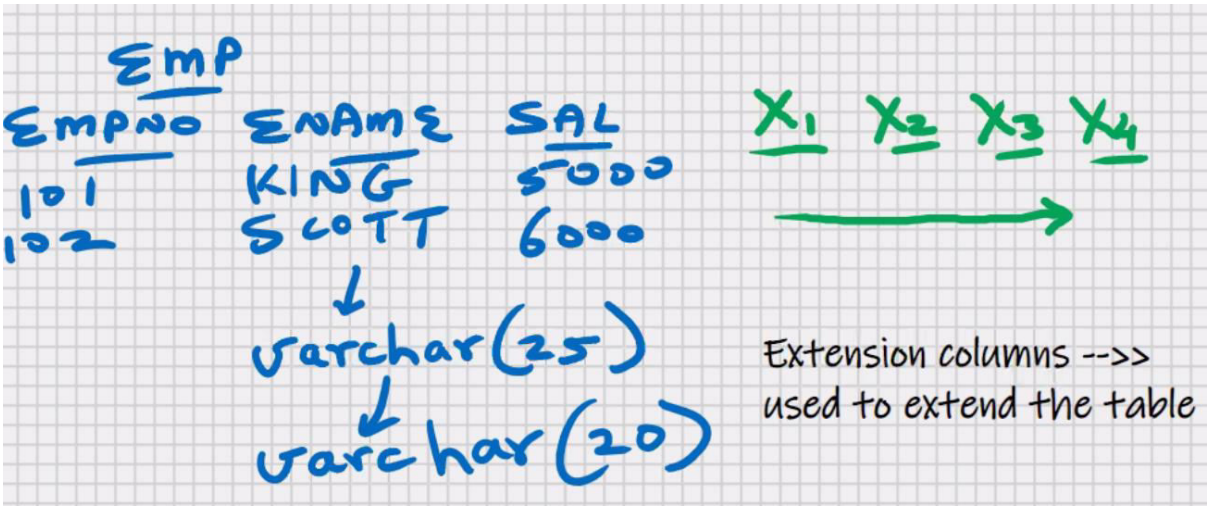
Alter table emp modify ename varchar (20);

/*data testing with x column*/

Update emp set ename = x;

Alter table emp drop column x;

- In oracle, You can reduce the width provided the contents are null
- Above procedure is recommended for MySQL also
- You must have few extra columns in every tables(**Implement in projects**)



- Change the datatypes of column

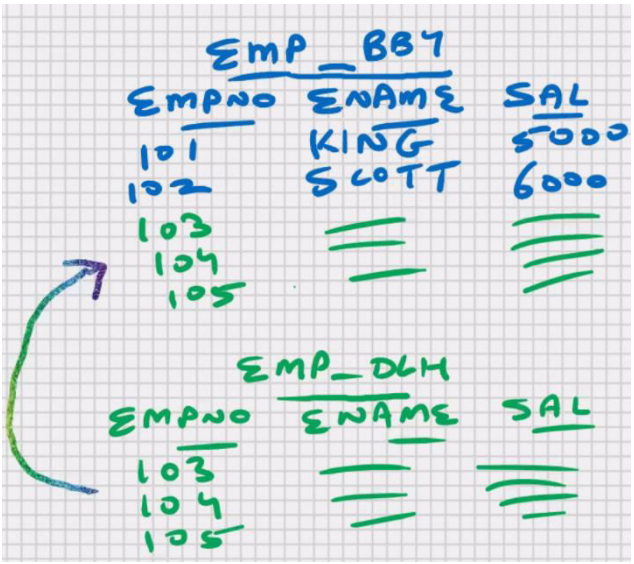
Change data type of empno from int to char(4)

Alter table emp add x int;
Update emp set x = empno, empno = null
Update emp set empno = null;
Alter table emp modify empno char (4);
/*data testing with x column*/
Update emp set empno = null;
Alter table emp drop column x;

- In Oracle, you can change the datatype provided the contents are null
- Used above method for MySQL also

- Copy the rows from one table into another table

Insert into emp_bby select * from emp_dlh;

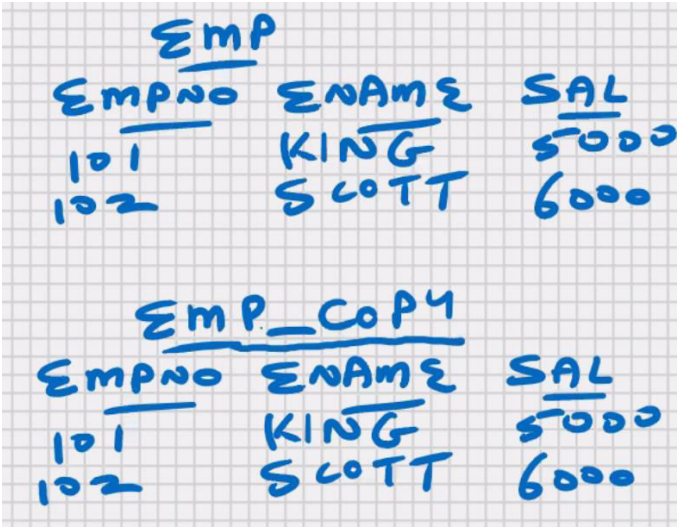


To copy certain rows only:-

Insert into emp_bby select * from Emp_dlh where ……;

- Copy a table (for testing purposes)

create table emp_copy
as
select * from emp;



To copy certain columns only:

Create table emp_copy
As
Select emp, ename from emp;

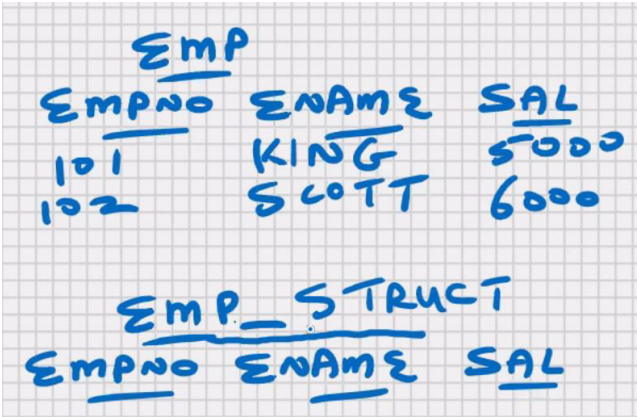
To copy certain rows only:

create table emp_copy
as
select * from emp where ……..;

- When you create a table using sub-query, the indexes on original table are not copy into the new table
- If you want the same indexes on the new table, you will have to create them manually

- When you create a table using subquery the constraints on original table are not copied into the new table, except for the not null constraint, because nullability is a feature of the datatype

- Copy only the structure of table



Method 1:-

```
Create table emp_struct
As
Select * from emp;
Delete from emp_struct;
Commit;
```

Method 2:-

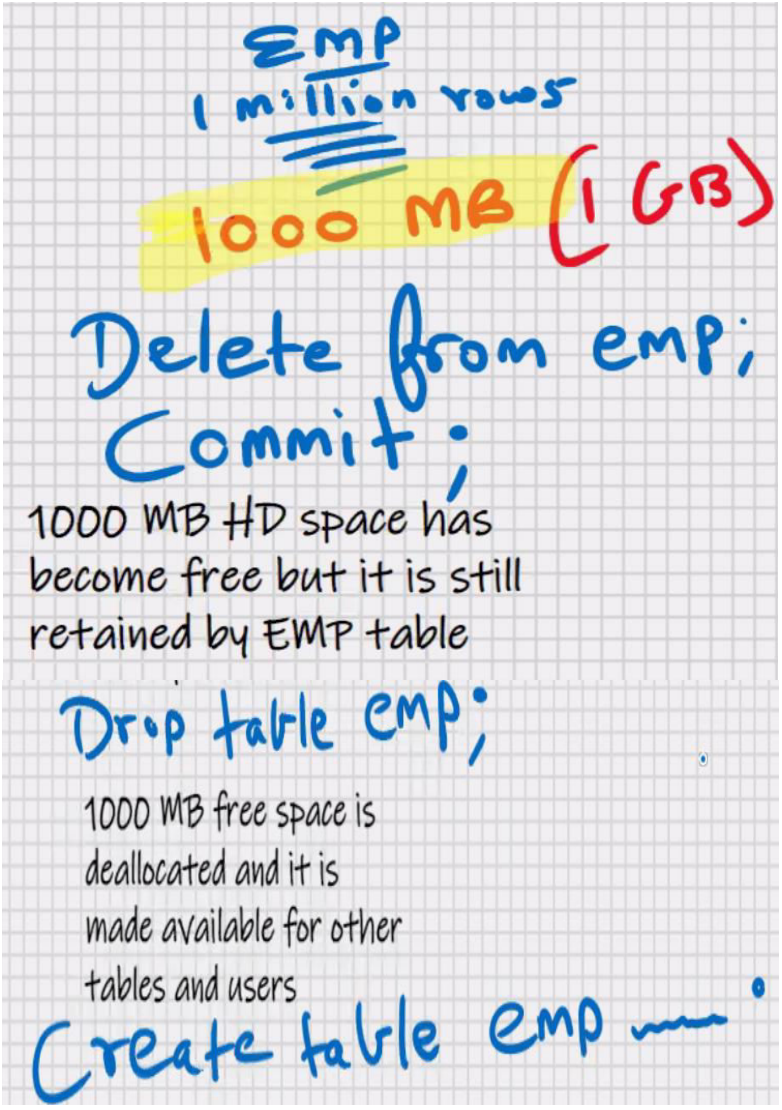
```
Create table emp_struct
As
Select * from emp;
Truncate table emp_struct;
```

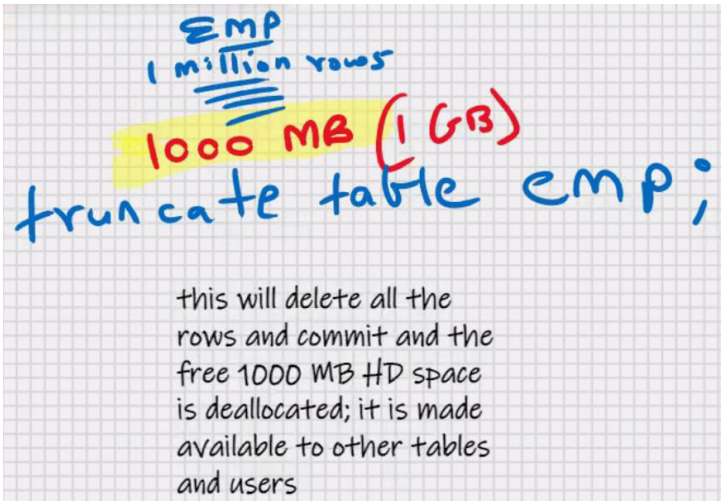
-> will delete all the rows and commit

Also

Difference between DELETE and TRUNCATE

DELETE	TRUNCATE
DML command	DDL command
Requires commit	Auto commit
Rollback is possible	Rollback is not possible
ANSI SQL and common for all RDBMS	Extra in MySQL and Oracle RDBMS
You can specify WHERE clause with delete	WHERE clause is not supported
Delete triggers on table will execute	Delete triggers on table will not execute

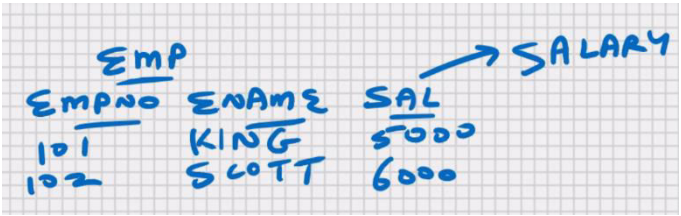




Method 3:-
Create table emp_struct
As
Select * from emp where 1 = 2; //give imposible condition

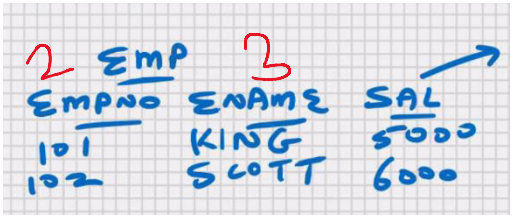
- Rename a column

create table emp_copy
as
select emp, ename,sal salary
from emp;
drop table emp;
rename table emp_copy to emp;



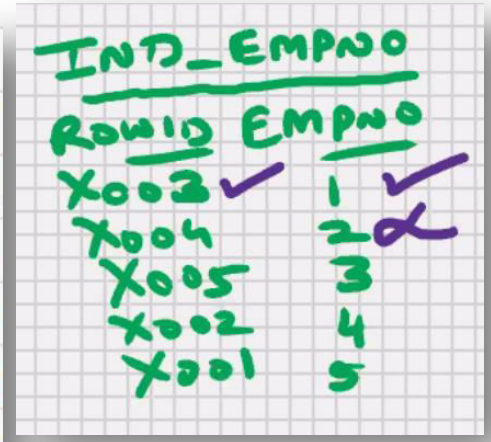
- Change the position of columns in table structure (because of null values) (for storage considerations)

```
create table emp_copy
as
select sal, empno ,ename from emp;
drop table emp;
rename table emp_copy to emp;
```



INDEXES

		EMP		
ROWID	EMPNO	ENAME	SAL	DEPTNO
X001	5	A	5000	1
X002	4	A	6000	1
X003	1	C	7000	1
X004	2	D	9000	2
X005	3	E	8000	2



- Present in all RDBMS, all DBMS, and some of the programming languages also
- Used to speed up the search operations (for faster access)
- Used to speed up the select statement with a where clause
- In MySQl and Oracle the indexes are automatically invoked as and when required.
- Indexes are automatically updated by MySQL and Oracle for all your DML Operations.
- MySQL and Oracle are self -managing RDBMS
- Duplicate values are stored in an index
- Null values are not stored in an index

  Select * from emp where empno is null;

  MySQL will do a full table scan

  Select * from emp where empno = 0;

In other RDBMS:-
Use index ind_empno;
Select * from emp where empno = 1;
In other RDBMS:
Insert/update/delete . . . . . :
REINDEX;

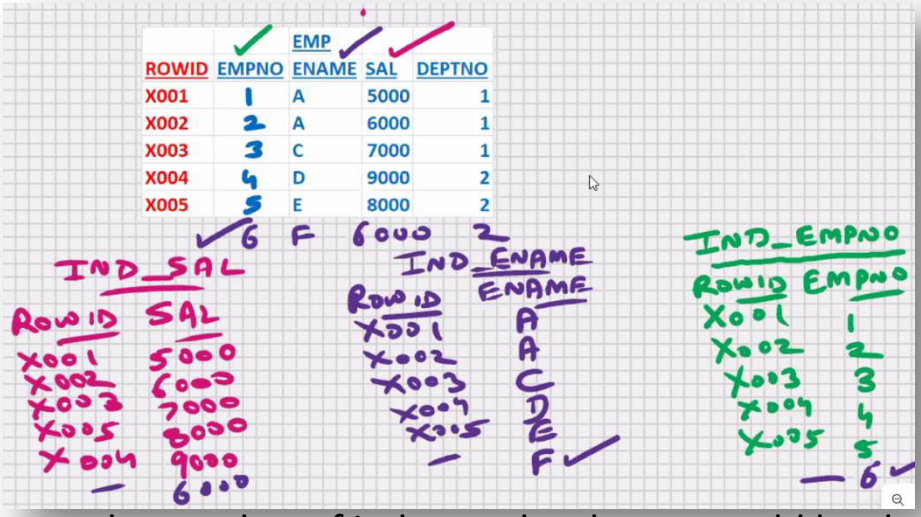
Select * from emp Where empno = 1;
Select * from emp Where ename = 'A';

EXECUTION PLAN -> plan created by MySQL as to how it's going to execute the SELECT statement

select * from emp
where empno = 1;
1. Read
2. Compile
3. Plan
4. Execute

Select * from emp Where sal > 7000 ;

- In MySQL and oracle , there no upper limit on the number of indexes per table



- Larger the number of indexes, the slower would be the DML operations
- Fast searching vs Fast DML operations? (AS per User-requirements)

  Select * from emp
  Where empno = 2;

  Select * from emp
  Where sal > 5000;

Select * from emp
Where empno= 2 and sal >5000;

- If you have 2 or more independent columns in the where clause create indexes for all such columns; and MySQL will use all the necessary indexex as and when required

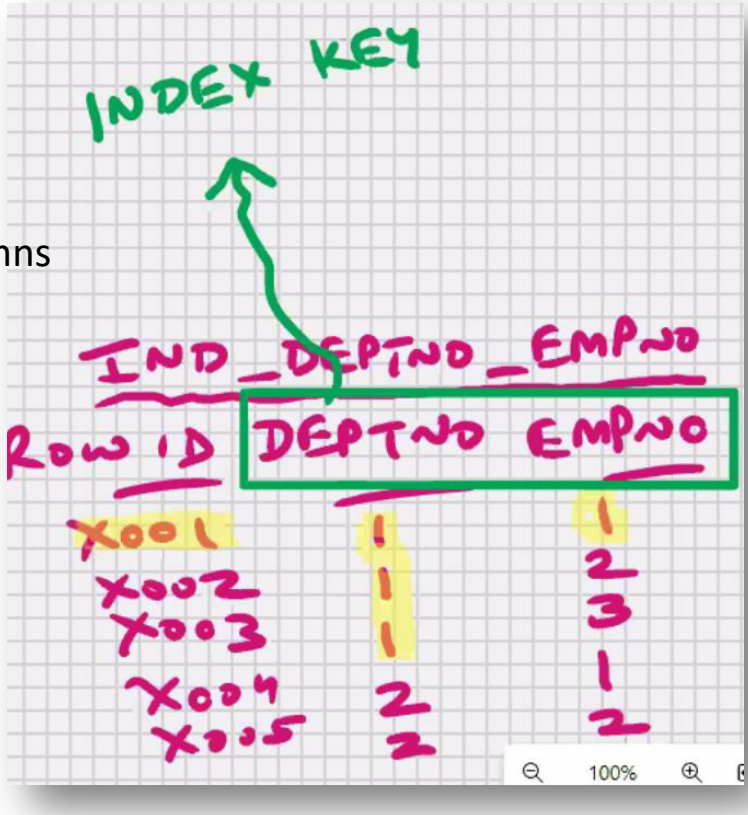
EMP

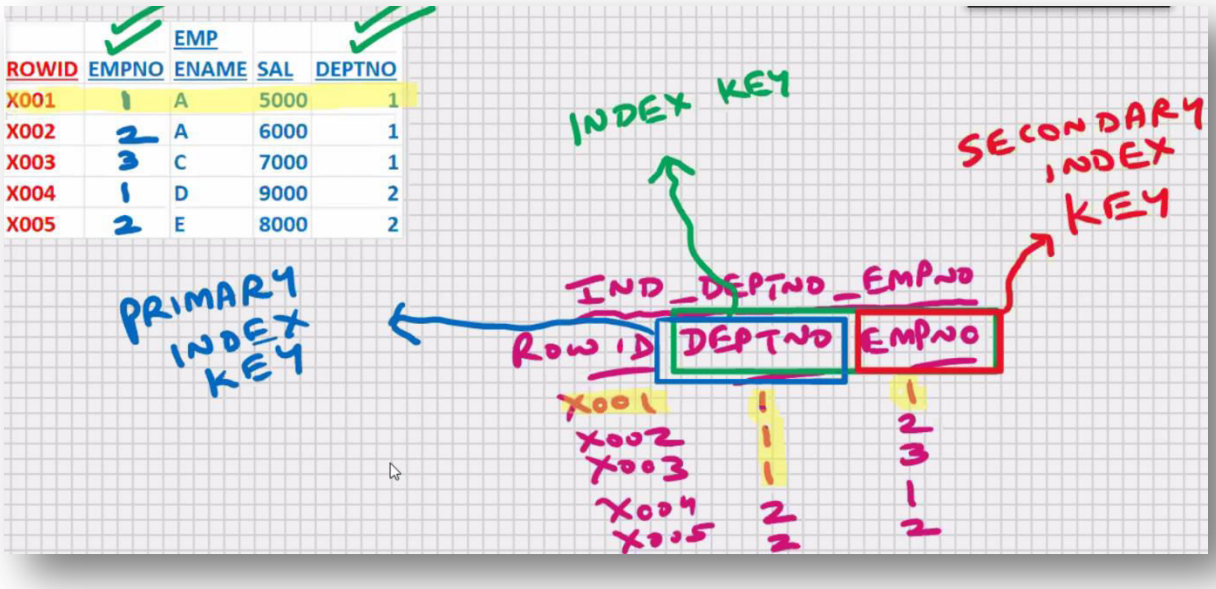
ROWID	EMPNO	ENAME	SAL	DEPTNO
X001	1	A	5000	1
X002	2	A	6000	1
X003	3	C	7000	1
X004	1	D	9000	2
X005	2	E	8000	2

Composite index -> combine 2 or more inter dependent columns in a single index

Select * from emp
Where deptno = 1 and empno = 1;

Index key -> column or set of columns
On whose basis the index has been
Created





- In **MySQL** you can **combine upto 32 columns** in a composite index and in oracle upper limit is 16 columns
- You cannot index text and blob

Select * from emp
where empnp = 1;

condition when an index should be created:

- Used to speed up the select statement with a where clause, order by clause, distinct, group by clause, union, intersect, minus
- If your select statements retrieves < 25% of the table data

Select * from emp where empno = 1;
Select * from emp Order by empno;
Select distinct empno from emp;
Select deptno, sum(sal) from emp Group by deptno;
Select * from emp where empno = 1;
Select * from emp where empno = 5;
Select * from emp where empno != 1;

- Key is another name for column
- Primary key and unique columns should always be indexed
- Primary key column is the best column for searching

EMP

ROWID	EMPNO	ENAME	SAL	DEPTNO
X001	1	A	5000	1
X002	2	A	6000	1
X003	3	C	7000	1
X004	4	D	9000	2
X005	5	E	8000	2

DEPT

ROWID	DEPTNO	DNAME	LOC
Y011	1	TRN	Bby
Y012	2	EXP	Dlh
Y013	3	MKTG	Cal

Select dname, ename from emp, dept
Where dept.deptno = emp.deptno;

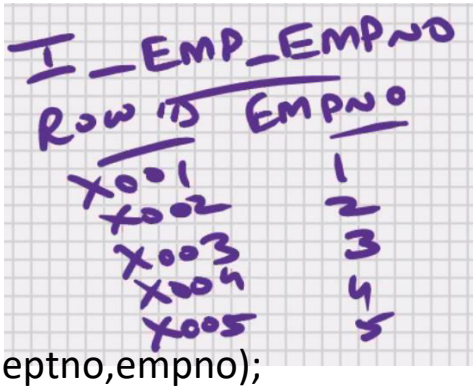
- Common column In join operations should always be indexed

In MySQL :

Create index indexname on table (column)

Create index i_emp_empno on emp(empno);
Select * from emp where empno = 1;

Select * from i_emp_empno; -> error



Create index i_emp_ename on emp(ename);
Create index i_emp_sal on emp(sal);

Create index i_emp_deptno_empno on emp(deptno,empno);
Create index i_emp_empno on emp(empno desc);//descendingorder

Create index i_order_onum on orders(onum desc);
Create index i_emp_deptno_empno on emp(deptno desc,empno);

Create index i_emp_deptno_empno
on emp(deptno desc,empno desc);

To see which all indexes are created for specific tables:-
show indexes from emp;
To see all indexes on all tables in all schemas:-
use information_schema;
select * from statistics; <- this is a system table
To drop the index:-
Drop index i_emp_empno on emp;

Unique index:
Create unique index i_emp_empno on emp(empno);

- performs one extra function, it wont allow the user to insert duplicate values in empno

- At the time of creating the unique index, if you already have duplicate values in empno, then MySQL will not allow you to create the unique index

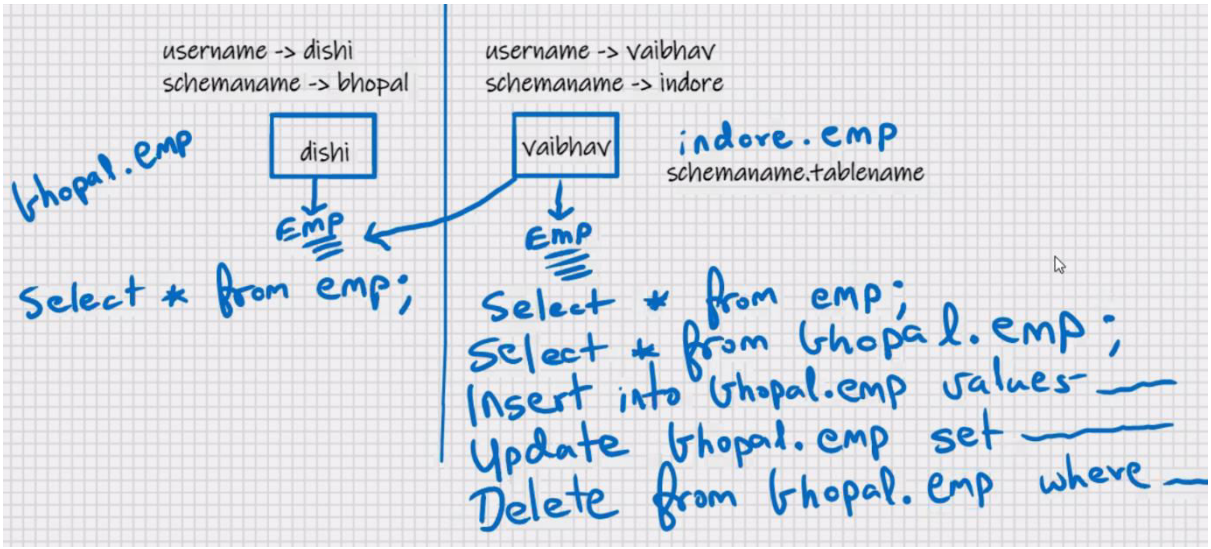
Types of Indexex:
1.Normal Index
2.Unique Index
3.bitmap Index
4.Index Organized table
5.index partitioning (global and Local)
6.etc

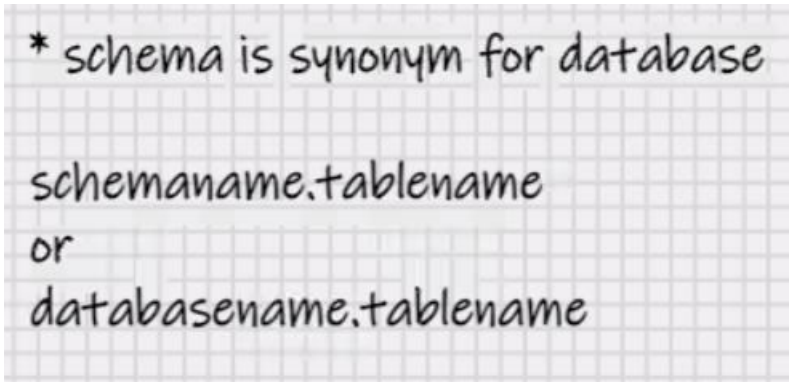
MySQL – SQL – PRIVILEGES

GRANT/REVOKE (DCL COMMANDS)

```
Dishi_mysql> grant select on emp to vaibhai;
Dishi_mysql> grant insert on emp to vaibhai;
Dishi_mysql> grant update on emp to vaibhai;
Dishi_mysql> grant delete on emp to vaibhai;
Dishi_mysql> grant select,insert on emp to vaibhai;
Dishi_mysql> grant all on emp to vaibhai;

Dishi_mysql> grant select on emp to vaibhai, manoj;
Dishi_mysql> grant select on emp to public;  //all users

Revoke select on emp from Vaibhav;
```

To see which all permissions have been granted and received:
Select * from information_schema.table_privileges;





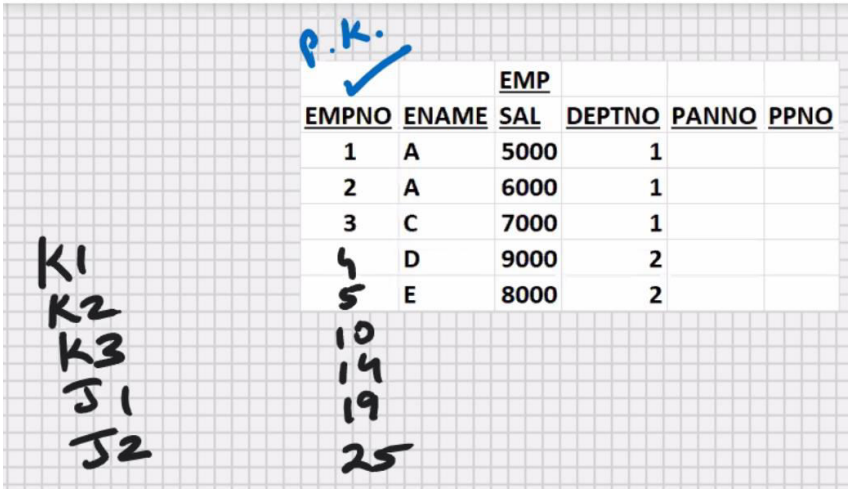
Dishi_mysql> grant select on emp to Vaibhav with grant option;
Vaibhav mysql> grant select on Bhopal.emp to manoj;

MySQL- SQL- CONSTRAINTS

- Limitation / restrictions imposed on a table

Primary Key (Primary column)

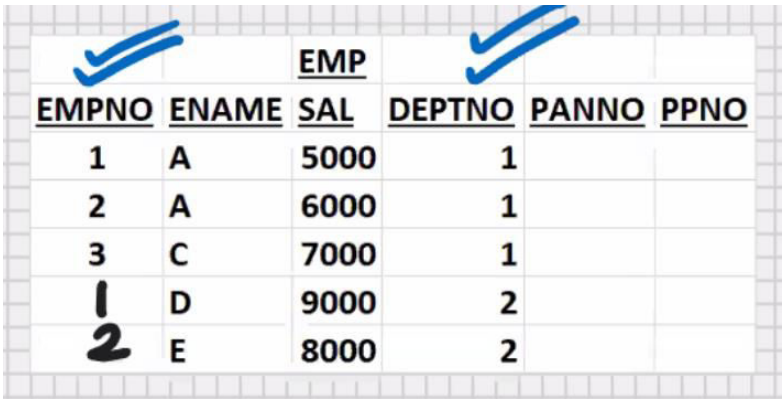
- Column or set of set of column that uniquely identifies a row(e.g. empno)
- Duplicate values are not allowed
- Null values are not allowed (mandatory column)
- Its recommended that every table should have a primary key



- Purpose of primary key is row uniqueness
- With the help of primary key you can distinguish between 2 rows of a table
- Text and blob cannot be primary key
- Unique index is automatically created

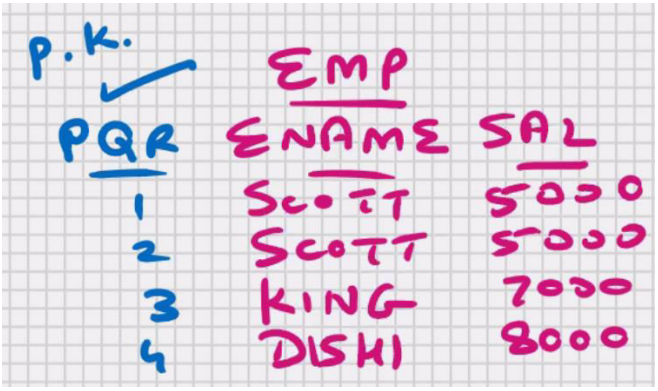
COMPOSITE PRIMARY KEY -> combine 2 or more columns together to serve the purpose of primary key

- In MySQL you can combine upto 32 columns in a composite primary key

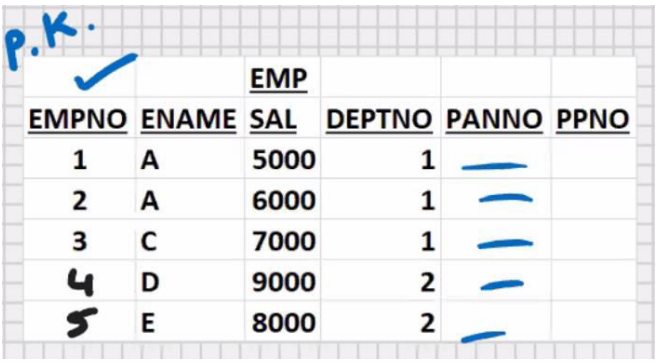


SURROGATE KEY ->

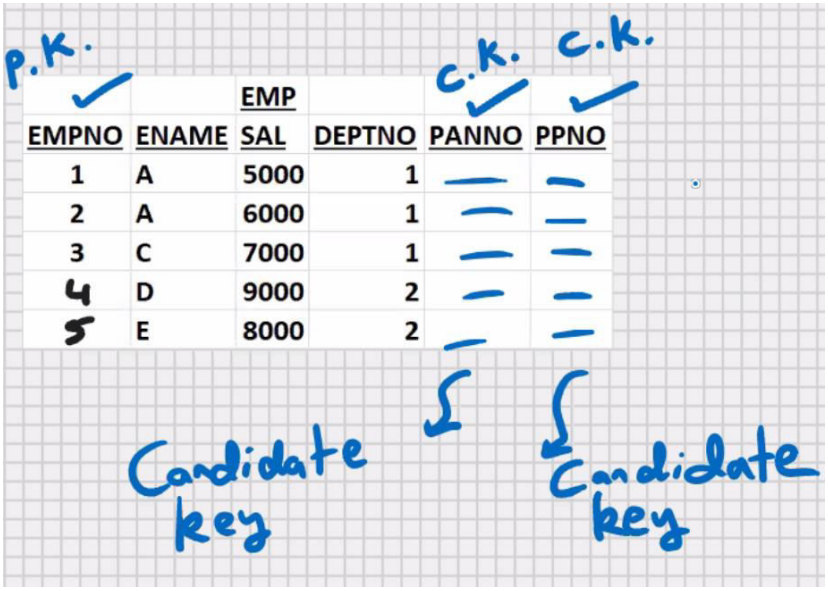
- If you cannot identify primary key in your table , add an extra column to the table, and make it the primary key of your table
- such a primary key which is not an original column table, such a primary key is known as surrogate key
- surrogate key is not a constraint
- surrogate key is a definition
- primary key column in the best column for searching and with char datatype the searching a retrieval is very fast
- for (surrogate key) primary key, char datatype is recommended



- if you declare a composite primary key, then mysql automatically creates a composite unique index
- YOU CAN HAVE ONLY 1 PRIMARY KEY PER TABLE



CANDIDATE KEY ->



- Candidate key is not a constraint
- Candidate key is a definition
- Besides the primary key, any other column in the table that can also serve the purpose of primary key, is a good candidate for primary key, is known as candidate key.
- Its good to have a couple of candidate keys in your table; in future if you drop the primary key (empno)column, then your table is left without a primary key; then you can immediately make PANNO or PPNO as the new primary key

```
Create table emp
(
Empno char(4) primary key,
Ename varchar(25),
Sal float,
Deptno int
);

Insert into emp values('5','F',5000,2);        -> error
Insert into emp values(null,'F',5000,2);        -> error
```

- All constraints are at server level
- You can insert using MySQL command line client, MySQL Workbench, MySQL – PL program, java, MS .Net, etc.; the constraints will always be valid
- You can perform DML operations using any front-end s/w, the constraints will always be valid
- This is known as DATA INTEGRITY
- Internally a constraint is a mysql created function, it performs the validations

To find out which all constraints you have created:-
Select * from information_schema.table_constraints;

Select * from information_schema.table_constraints;
Where table_schema = 'bhopal';

Select * from information_schema.key_column_usage
Where table_name = 'emp';

- Unique index is automatically created

Show indexes from emp;

To drop the primary key constraint:-
Alter table emp drop primary key;

To add the primary key afterwards to an existing table:-
Alter table emp add primary key (empno);

To change the primary key column:-

1. drop the original primary key constraint
2. add a new primary key constraint

Create table emp
(
Empno char(4) **primary key**,
Ename varchar(25),
Sal float,
Deptno int,
Primary key (deptno, empno) //(parent,child)
);

- If you drop the constraint, then the index is dropped automatically

Alter table emp drop primary key;

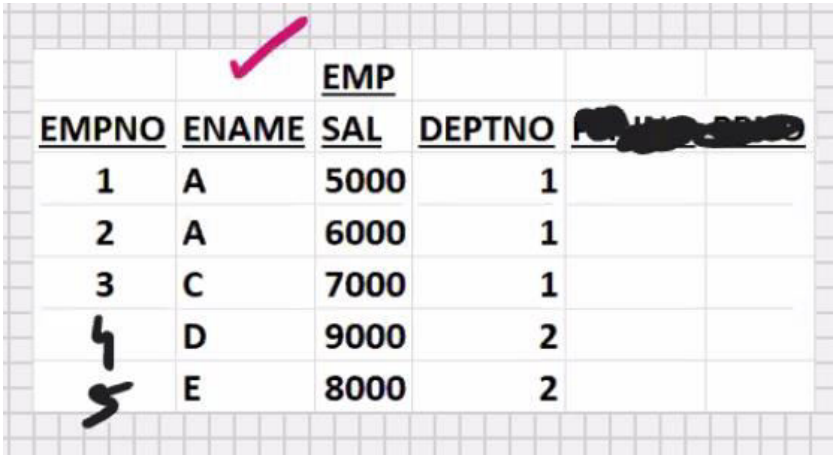
To add primary key afterwards to an existing table:
Alter table emp add primary key(deptno, empno);
To change the primary key column:-

1. Drop the original primary key constraint
2. Add a new primary key

Constrains are of 2 types:-
a.column level constraint (specified on single column)
b.table level constraint (composite) (specified on combination of 2 or more columns) (has to be specified at the end of the table structure)

NOT NULL CONSTRAINT

- Null values are not allowed (it's a mandatory column) (similar to primary key)
- Duplicate values are allowed (unlike primary key)



EMP

EMPNO	ENAME	SAL	DEPTNO	[redacted]
1	A	5000	1	
2	A	6000	1	
3	C	7000	1	
4	D	9000	2	
5	E	8000	2	

- Can have any number of not null constraints per table (unlike primary key)
- You cannot have composite not null constraint (unlike primary key)
- Always a column level constraint (cannot have table level not null constraint)

Create table emp
(
Empno char(4) **primary key**,
Ename varchar(25) not null,
Sal float not null,
Deptno int
);

- In MySQL, nullability is a part of the datatype

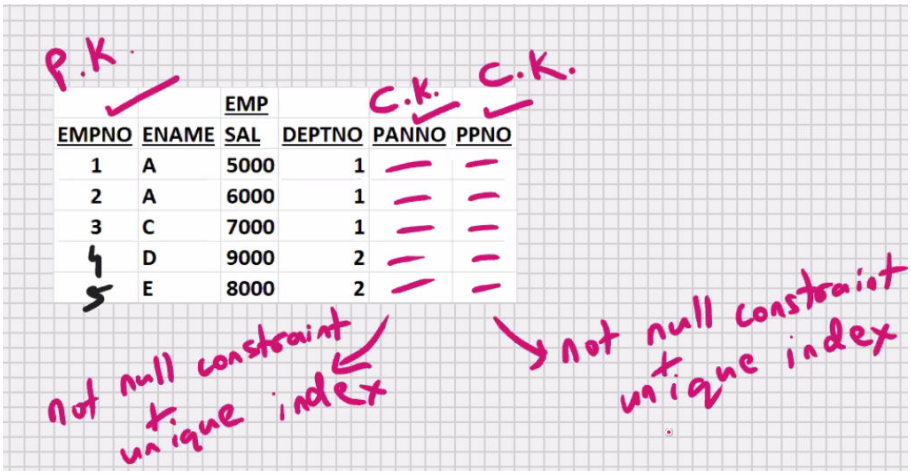
To find out which are the not null columns:-
Desc emp;

To remove the not null constraint afterwards:-
Alter table emp modify ename varchar(25) null;

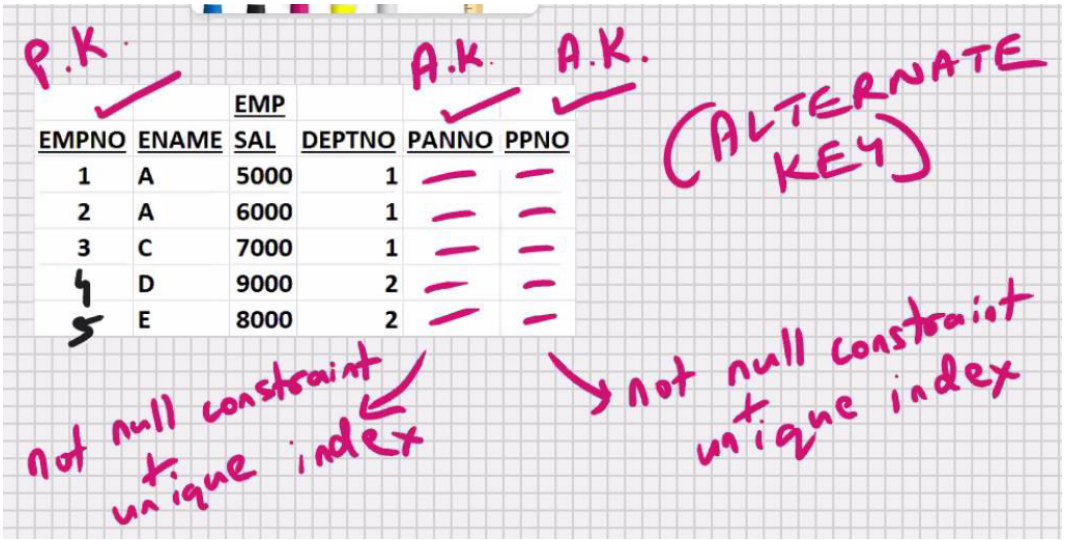
To add the not null constraint afterwards for an existing table:-
Alter table emp modify ename varchar(25) not null;

Solution for candidate key column:-

Not null constraint + unique index

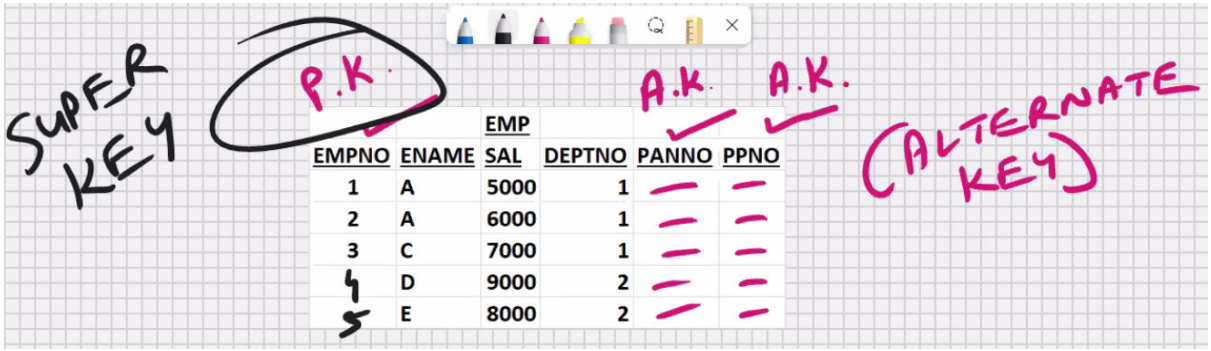


- With the help of above , indirectly, you can have multiple primary keys in the table
- Alternate key is not a constraint



- Alternate key is a definition
- Alternate key for the candidate key column, if you specify a not null constraint and create an unique index, then it becomes an alternative to primary key, then such a candidate key column is known as **alternate key**
- **Super key :** is not a constraint
- **Super key** is a definition

- Super key if you have an alternate key in the table, then primary key is known as super key

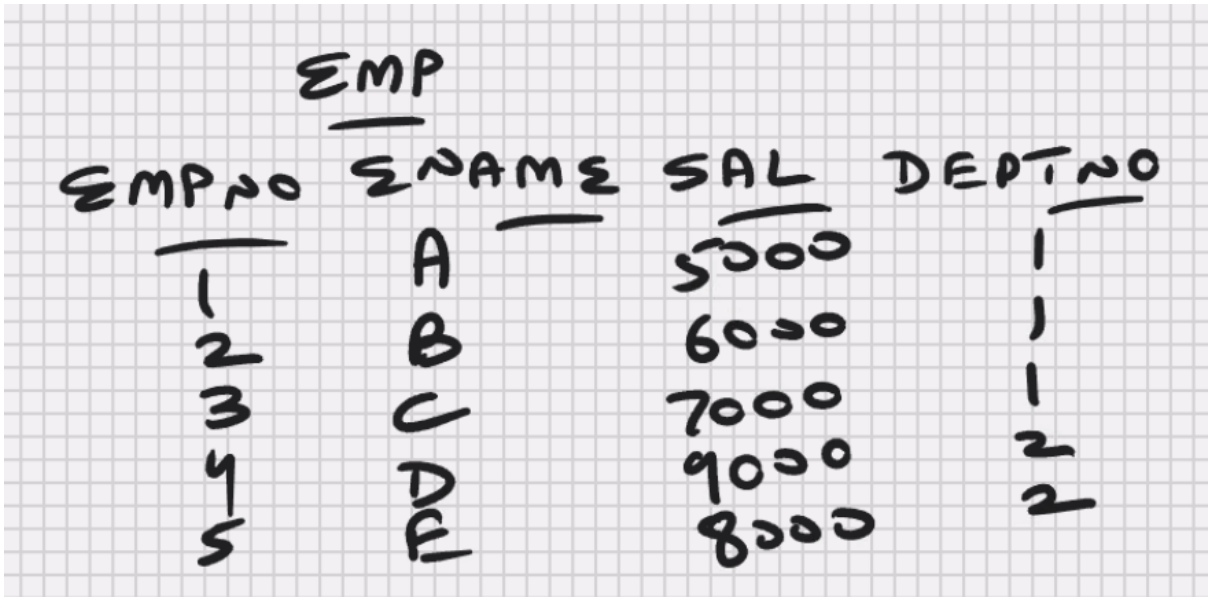


EMP

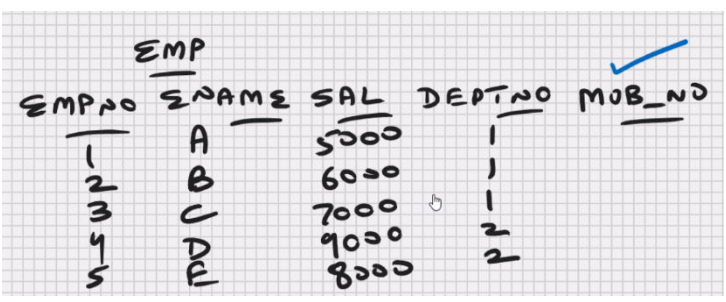
EMPNO	ENAME	SAL	DEPTNO	PANNO	PPNO
1	A	5000	1	—	—
2	A	6000	1	—	—
3	C	7000	1	—	—
4	D	9000	2	—	—
5	E	8000	2	—	—

Unique constraint

EMP table



- Duplicate values are not allowed (it has to be unique) (similar to primary key)
- Null values are allowed (unlike key)
- You can enter any number of null values
- Unique index is automatically created (similar to primary key)
- Text and blob cannot be unique (similar to primary key)



- In MySQL, you can combine upto 32 columns in a composite unique
- **You can have any number of unique constraints per table (unlike primary key)**

Create table emp
(
Empno char(4) **,**
Ename varchar(25),
Sal float,
Deptno int,
Mob_no char(15) unique, -> column level constraint
Unique (deptno, empno) -> table level constriant
);

```
select * from information_schema.table_constraints;

select * from information_schema.table_constraints
where table_schema = 'bhopal';

select * from information_schema.key_column_usage
where table_name = 'emp';
```

- Unique index automatically created (similar to primary key)

Show indexes from emp;

- Unique constraint is also an index so to drop it:-

Drop index mob_no on emp;

Drop index deptno on emp;

- To add unique constraint afterwards:-
  Alter table emp

Add constraint u_emp_mob_no unique (mob_no);
Constraint u_emp_mob_no -> optional
Select * from information_schema.table_constraints
Where table_schema = ‘bhopal’;

```
Create table emp
(
Empno char(4) ,
Ename varchar(25),
Sal float,
Deptno int,
Mob_no char(15) ,
Unique (deptno, empno) ,
Unique (mob_no)
);
```

- Column level constraint can be specified at table level (at the end of structure), but a table level composite at table level (at the end of structure), except for the not null constraint which is always a column levelconstraint, and therefore the syntax will not support from specifying it at table level

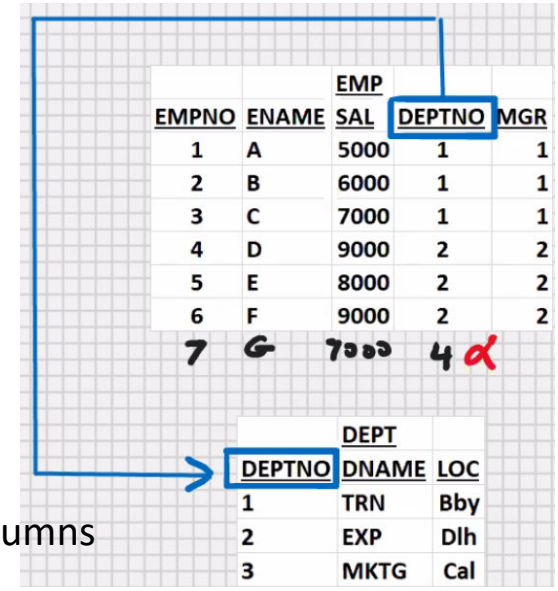
FOREIGN KEY

EMP

EMPNO	ENAME	SAL	DEPTNO	MGR
1	A	5000	1	1
2	B	6000	1	1
3	C	7000	1	1
4	D	9000	2	2
5	E	8000	2	2
6	F	9000	2	2

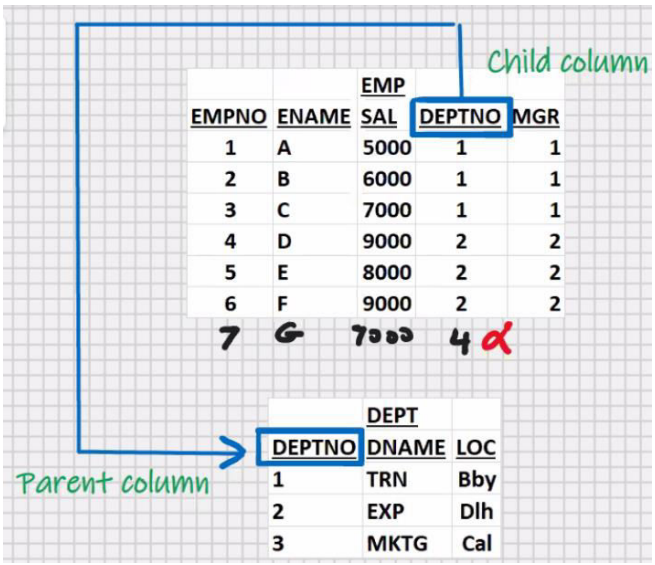
DEPT

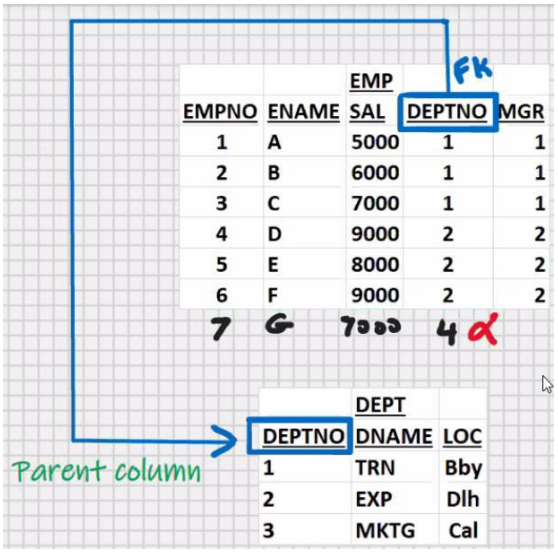
DEPTNO	DNAME	LOC
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal



Foreign key is also known as foreign column

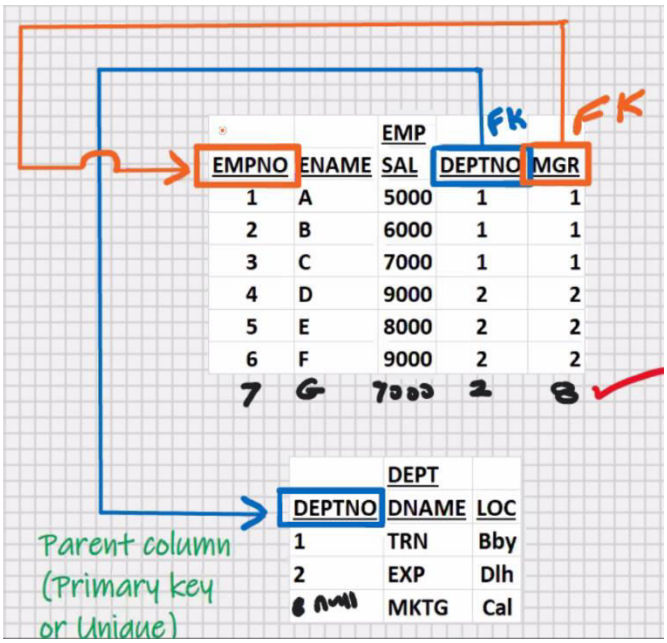
- Column or set of columns that References a column or set of columns of some table
- foreign key constraint is not required to write a join
- foreign key constraint is specified on the child column (not the parent column)

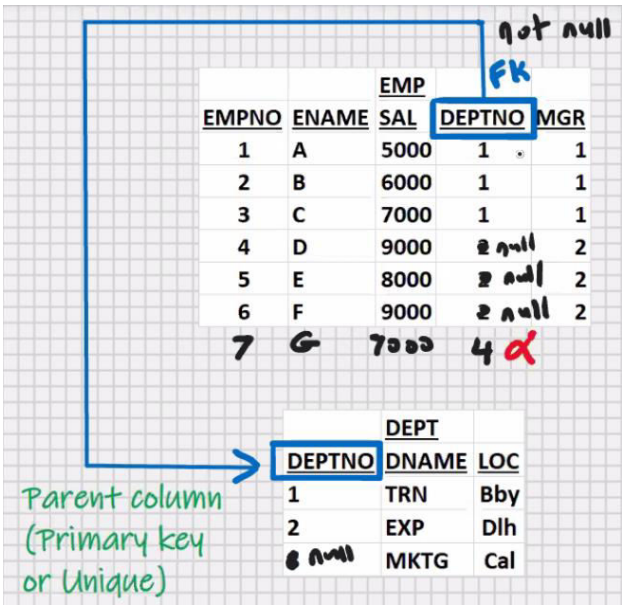




- parent column has to be primary key or unique (this is a prerequisite for foreign key)
- foreign key will allow duplicate values (unless specified otherwise)

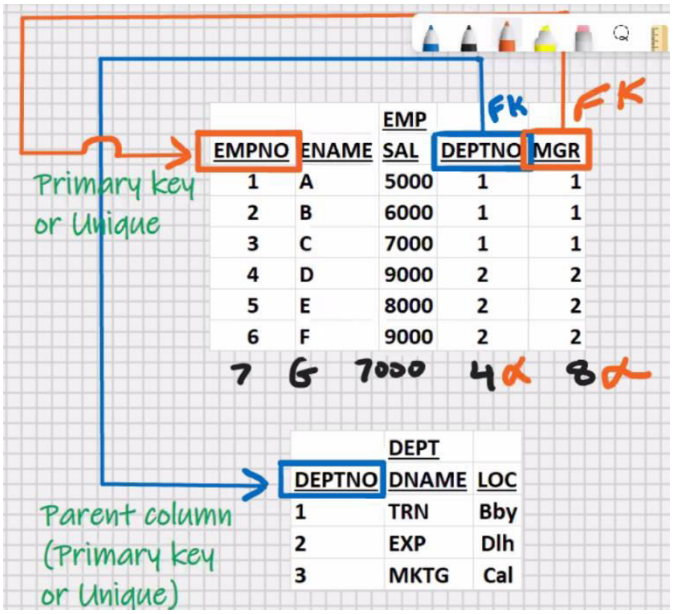
- foreign key will allow null values also (unless specified otherwise)





- foreign key may reference column of same table also (known as self-referencing)

```
create table dept
(
Deptno int primary key,
Dname varchar(15),
Loc varchar(10)
);

Create table emp
(empno char(4) primary key,
Ename varchar(25),
Sal float,
Deptno int,
Mgr char(4),
Constraint fk_emp_deptno foreign key(deptno)
References dept(deptno),
```



Constraint fk_emp_mgr foreign key(mgr)
References emp (empno)
);

Constraint fk_emp_deptno -> optional
Constraint fk_emp_mgr -> optional

Select * from information_schema.table_constraints;
Select * from information_schema.table_constraints
Where table_schema = ‘bhopal’;

Select * from information_schema.key_column_usage
Where table_name = ‘emp’;

Alter table emp drop foreign key fk_emp_deptno;

To add the constraint afterwards:

Alter table emp add constraint fk_emp_deptno foreign key(deptno)
references dept(deptno);

Delete from dept where deptno = 3;

- you can delete the parent row provided child rows don’t exist

Delete from dept where deptno = 2;

- you cannot delete the parent row when child rows exist

but still if you want to delete then solution is:
1.delete from emp where deptno = 2;
2. delete from dept where deptno = 2;

```
Create table emp
(empno char(4) primary key,
Ename varchar(25),
Sal float,
Deptno int,
Mgr char(4),
Constraint fk_emp_deptno foreign key(deptno)
References dept(deptno) on delete cascade,
Constraint fk_emp_mgr foreign key(mgr)
References emp (empno)
);
```

On delete cascade -> if you delete the parent row, then MySQL will automatically delete the child rows also

To preserve the child rows :

```
Update emp set deptno = null where deptno = 2;
Delete from dept where deptno = 2;
```

- you can update the parent column provided child rows don't exist

```
Update dept set deptno = 4 where deptno = 3;
```

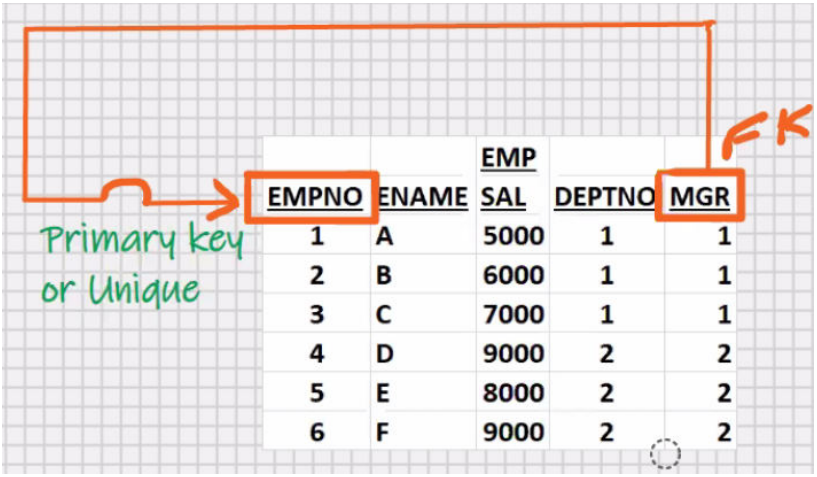
- you cannot update the parent column when child column is exist

```
update dept set deptno = 4 where deptno = 2;  -> error
Create table emp
(empno char(4) primary key,
Ename varchar(25),
Sal float,
Deptno int,
Mgr char(4),
Constraint fk_emp_deptno foreign key(deptno)
```

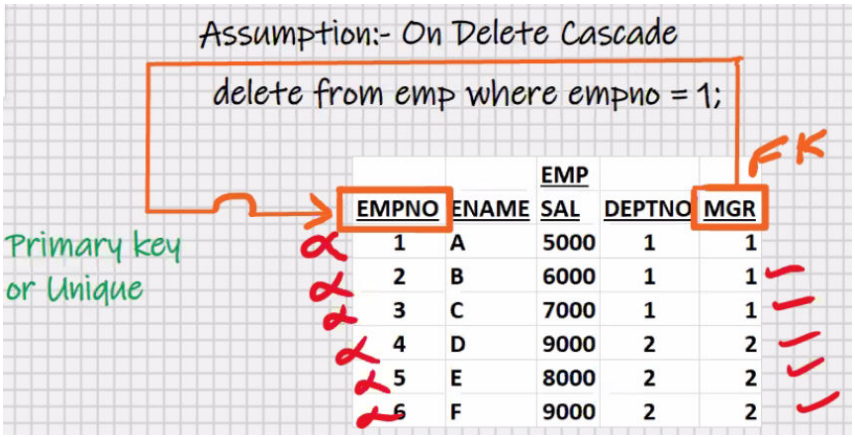
References dept(deptno) **on delete cascade on update cascade**,
Constraint fk_emp_mgr foreign key(mgr)
References emp (empno)
);

On update cascade -> if you update the parent column, then the child rows are updated automatically

Update dept set deptno = 4 where deptno = 2;

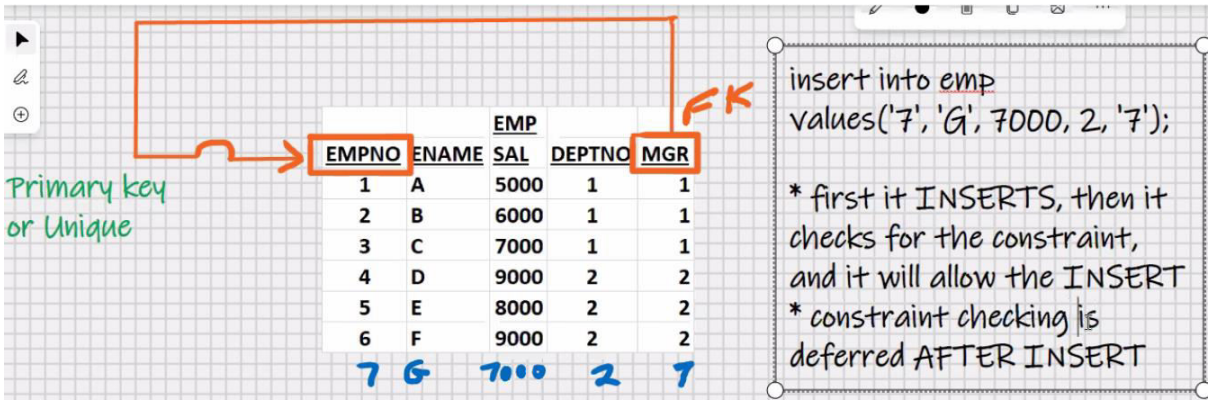


EMP				
EMPNO	ENAME	SAL	DEPTNO	MGR
1	A	5000	1	1
2	B	6000	1	1
3	C	7000	1	1
4	D	9000	2	2
5	E	8000	2	2
6	F	9000	2	2



EMP				
EMPNO	ENAME	SAL	DEPTNO	MGR
1	A	5000	1	1
2	B	6000	1	1
3	C	7000	1	1
4	D	9000	2	2
5	E	8000	2	2
6	F	9000	2	2

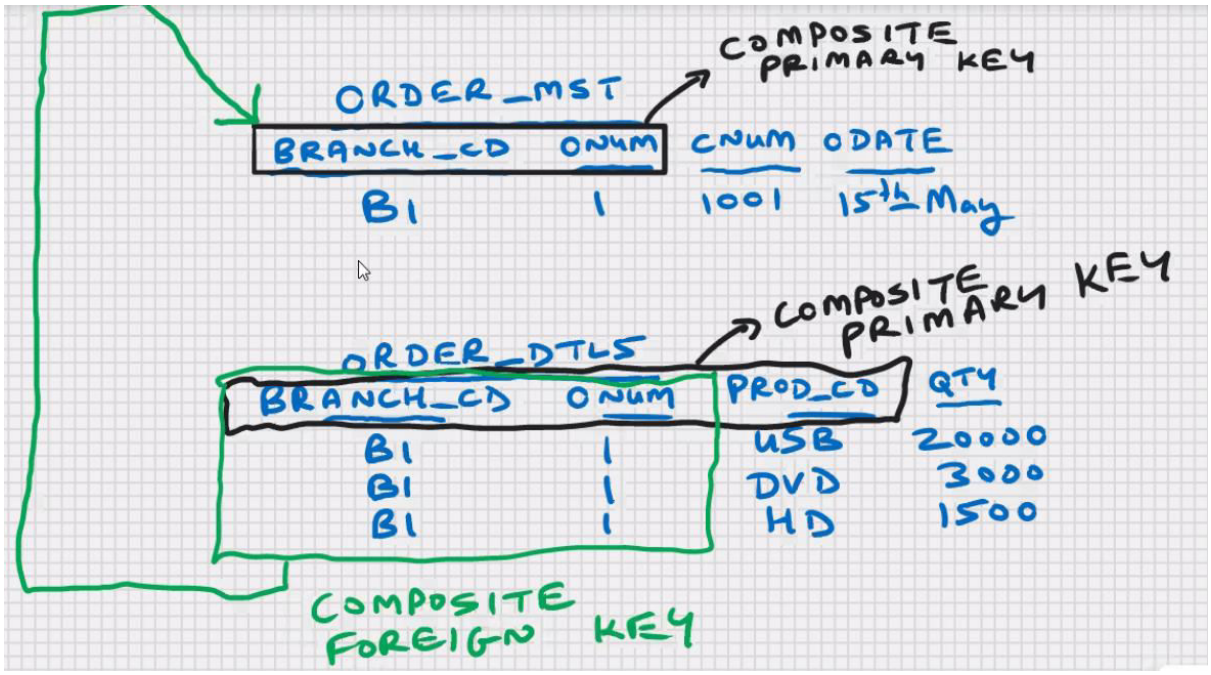
- avoid on delete cascade in the event of self referencing; you may delete more rows than expected



Insert into emp
Values ('7','G',7000,2,'8');

- first it inserts, then it checks for the constraint, then it will rollback and give an error message

COMPOSITE FOREIGN KEY



Create table order_mst …..;

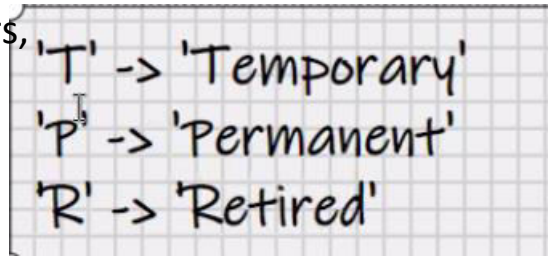
Create table order_dtls
(
Branch_cd char(4),

Onum int,
Prod_cd char (4),
Qty int,
Primary key(branch_cd, onum,prod_cd),
Foreign key(branch_cd,onum) references order_mst
(branch_cd,onum)

```
create table order_dtls
(
branch_cd char(4),
onum int,
prod_cd char(4),
qty int,
primary key(branch_cd,onum,prod_cd),
constraint fk_emp foreign key(branch_cd,onum)
references order_mst(branch_cd,onum) ON DELETE CASCADE
);
```

CHECK CONSTRAINT

- used for validation (used for checking purposes)
- e.g. sal < 10000 ,age >21,etc
- in check constraint you can use relational operators, logical operators,arithmetic operators, special operators(in, between, like), call single_row functions
  e.g. upper, round, etc.



create table emp
(empno int auto_increament primary key,
Ename varchar (25) check (ename = upper(ename)),
Sal float **default 7000**
check (sal > 5000 and sal < 300000),
Deptno int,
Status char(1) **default 'T'**
check (status in ('T','P','R')),

```
(sal between 5001 and 299999)
```

Comm flat not null,
Mob_no char(15) unique,
Check (sal+comm < 500000),
Constraint fk_emp foreign key(deptno) references dept (deptno)
);

- default is not a constraint
- default is a clause that you can use with create table
- if user is entering some, then it will take value
- if nothing is entered, then it will take default value
- to make use default value and auto increment, use the following insert statement:-

insert into emp (ename, deptno, comm, mob_no)
values (……..);

```
create table emp
(
empno int,
ename varchar(25) check(ename = upper(ename)),          <- column level
sal float check(sal between 5001 and 299999),          <- column level
deptno int,
status char(1) check(status in('T','P','R')),          <- column level
comm float,
mob_no char(15),
check(sal+comm < 500000)                          <- table level
);
```

Data is of 2 types:

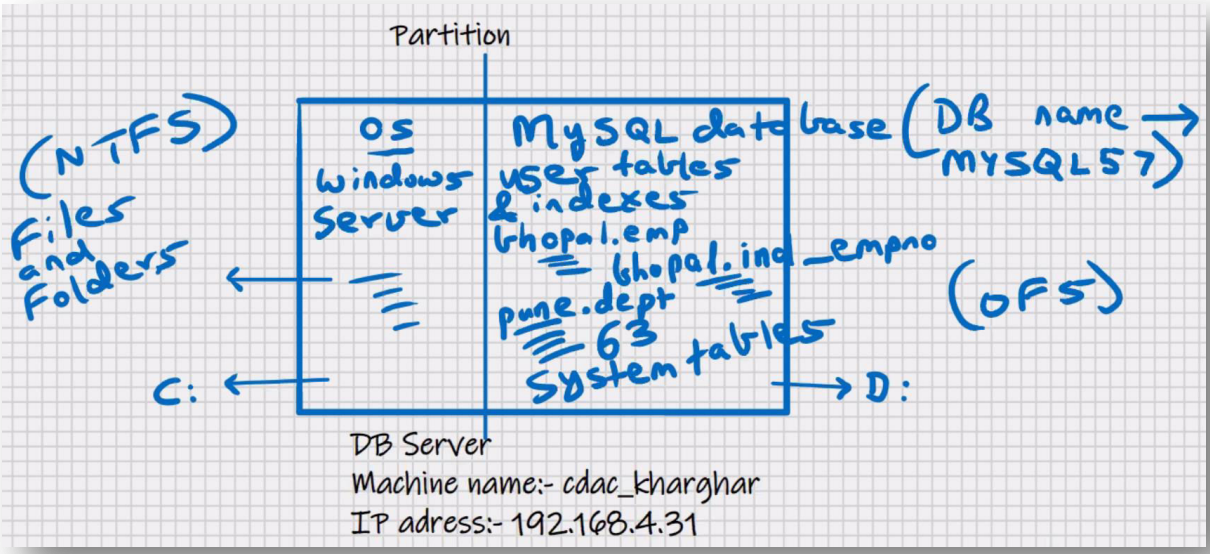
1. **<u>User data</u>**

- user created
- user tables and indexes

2. **<u>System data</u>**
    - System created (MySQL created)
    - Data that is stored in system tables
    - **System data** is also known as **METADATA**
    - Metadata -> data about data

SYSTEM TABLES

- 63 system tables (**they are views**) in MySQL
- Store complete information about the database
- System tables are stored in **information_schema**
- **Data dictionary ->** set of system tables is known as data dictionary
- **Also known as DATABASE CATALOG**
- E.g. statistics, table_constraints, key_column_usage, table_privileges, etc.
- All system tables are read_only
- You can only select from them (DML operations are not allowed)
- **DDL for you is DML for system tables**


Partition
(NTFS)
Files and Folders
OS
Windows Server
MySQL database
user tables & indexes
bhopal.emp
bhopal.ind_empno
pune.dept
63 System tables
(DB name → MYSQL57)
(OFS)
C:
D:
DB Server
Machine name:- cdac_kharghar
IP adress:- 192.168.4.31

OFS -> Oracle file system (based on linux kernel)

Oracle linux (known as unbreakable linux) -> most secure linux in the world (designed specifically to host the database)

MySQL – STORED OBJECTS

- Objects that are stored in the database
- E.g. tables, indexes

VIEWS (VERY IMPORTANT)

- Present in all RDBMS and some of the DBMS also

	EMP		
EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

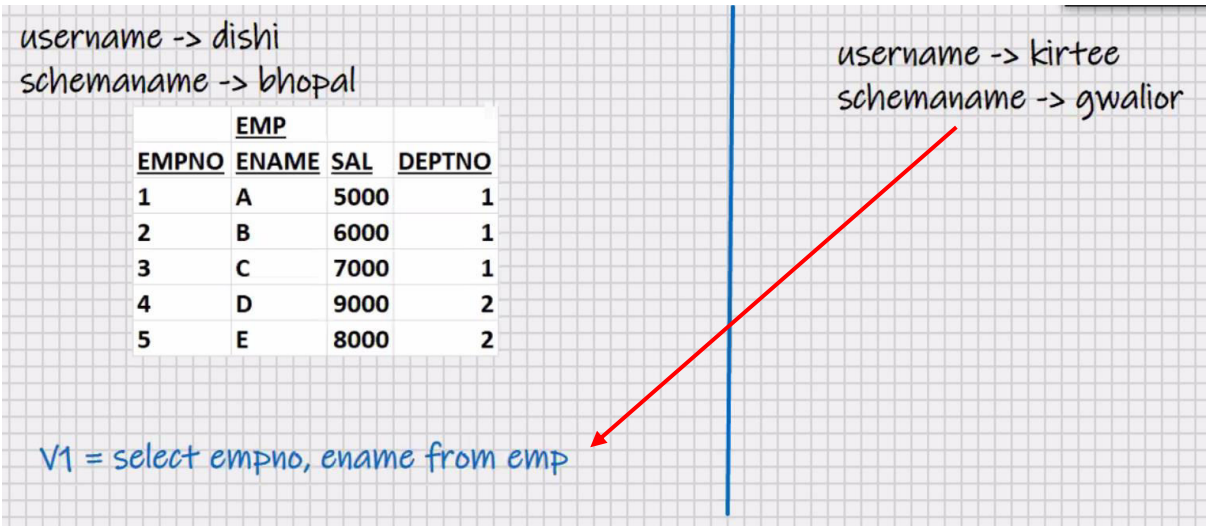
- View is a handle to a table
- View is a HD pointer(stores the address of table)
- View is known as locator (locator is a HD pointer)
- Used to restrict the access of users
- For **security purposes**
- Used for indirect access to the table

dishi_mysql > create view v1

as

select empno, ename from emp;

View created.



Dishi_mysql> grant select on v1 to kirtee;

Kirtee_mysql> select * from Bhopal.emp; -> error

Kirtee_mysql> select * from Bhopal.v1;

EMPNO	ENAME
1	A
2	B
3	C
4	D
5	E

- Used to restrict the column access
- View is a form of Encapsulation (form of data hiding)

Dishi_mysql> grant select on v1 to

kirtee,Vaibhav,manoj;

- View does not contain data
- Only the definition is stored, data Is not stored
- View is a stored query
- The select statement on which is based, is stored in a system table

- The select statement on which is based is stored in the DB in the compiled format
- View is an executable format of select statement
- Hence the execution is very fast
- Hiding the source code from end user

Kirtee_mysql> insert into Bhopal.v1 values(6,'F');

- DML operations can be performed on a view
- DML operations can be done on a view will affect the base table
- Constraints that are specified on the base table, they will be enforced when you INSERT via the view
- ENTIRE APPLICATION IS BUILT ON VIEWS

Dishi_mysql> drop view v1;

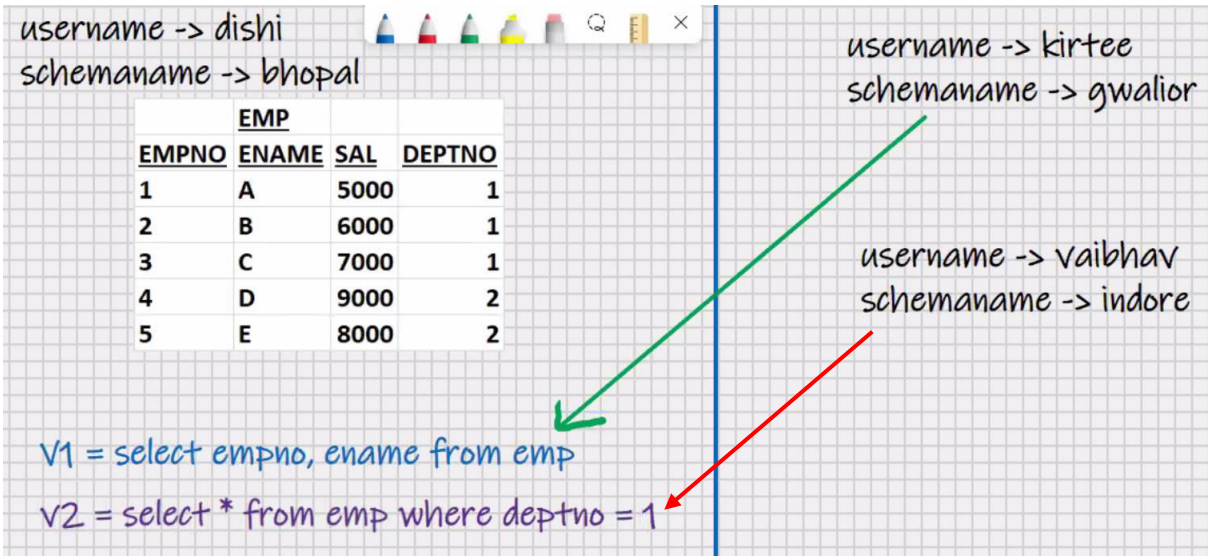
- Only owner can drop the view

Dishi_mysql> create view v2

As

Select * from emp where deptno = 1;

View created.



Dishi_mysql> grant select on v2 to Vaibhav;

Vaibhav_mysql> select * from Bhopal.v2;

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1

- Used to restrict the row access

Dishi_mysql> grant select, insert on v2 to Vaibhav;
Vaibhav_mysql> insert into Bhopal.v2 values(6, 'F',6000, **2)**;
->**allowed**

Dishi_mysql> create view v2
As
Select * from emp where deptno = 1 with check option;
Dishi_mysql> grant select, insert on v2 to Vaibhav;
Vaibhav_mysql> insert into Bhopal.v2 values(6,'F',6000,**2**); **<- error**

- View with check option is similar to check constraint
- Used to enforce different checks for different users

Dishi_mysql> Drop view v2;

Create view v1
As
Select empno, ename from emp;

To change the select statement in which the view is based:-
Create view v1 as ……. ;
OR
Create or replace view v1 as select ……. ;

Create or replace view v1
As
Select ename, sal from emp;

Select * from v1;

- View based on computed column, expression, function, etc
- You can only select from this view

- DML operations are not allowed

```
Create or replace view v1
As
Select upper(ename) u_ename, sal*12 annual from emp;
```

```
Create or replace view v1
As
Select deptno, sum(sal) from emp
Group by deptno;
```

- View based on **group by clause**
- You can only select from this view
- DML(insert,update,delete) operations are not allowed

```
Create or replace view v1
As
Select dname,ename  from emp,dept
Where dept.deptno = emp.deptno;
```

```
Select * from v1;
```

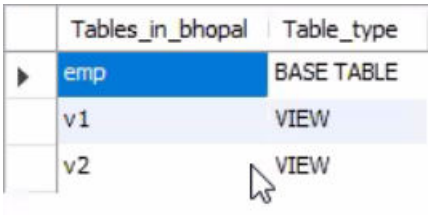
- View based on **join**
- You can only select from this view
- DML(insert,update,delete) operations are not allowed

- View based on view is allowed
- Uses

1. Union of>255 select statements
2. Function within function > 255 levels
3. Sub-queries > 255 levels
4. To simplify the writing of complex select statements
   e.g. join of 40 tables, etc.

- Complex queries can be stored in view definition

show tables; -> will show tables and views but it wont tell which is view
to see which is a table and which is a view
show full tables;



	Tables_in_bhopal	Table_type
▶	emp	BASE TABLE
	v1	VIEW
	v2	VIEW

to select statement on which the view is based:-
show create view v1;

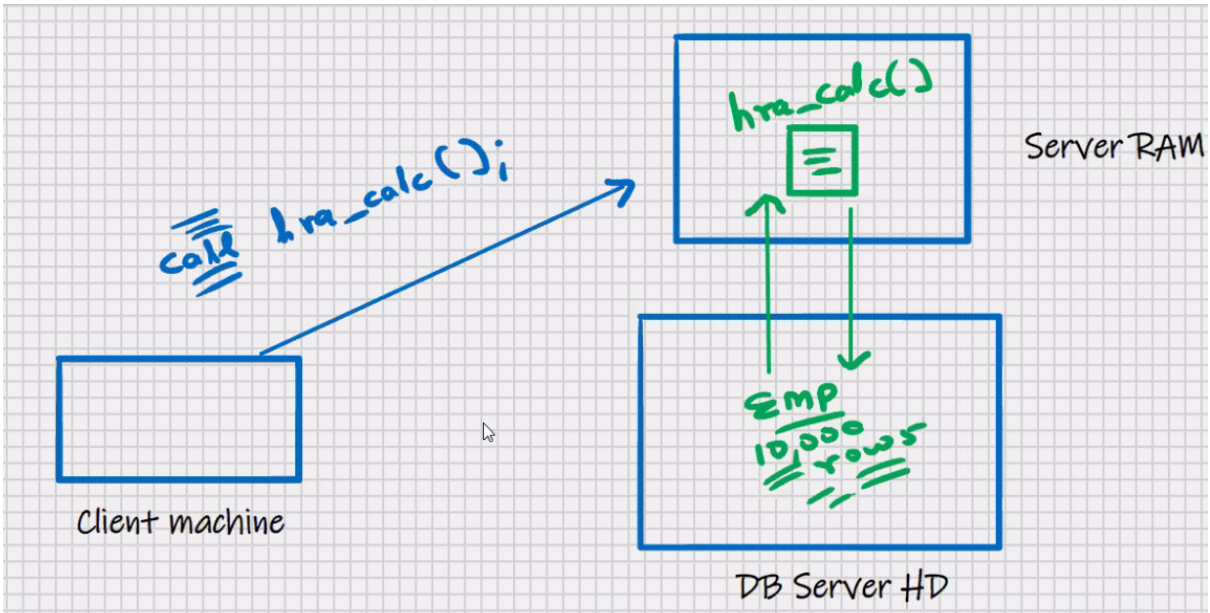
use information_schema;
show tables;
show full tables;
show create view character_set; //cant see source code -> error

Advanced features:-

- Types of views
- Uses:-
- To enforce relational methods on object tables
- To enforce object method on relational tables
- Convert 3D table into 2D table
- Convert 2D table into 3D table
- Migration
- Data mapping
- EAI (enterprise application integration)
- ElM (enterprise integration management)

MySQL -PL

- MySQL programming language
- Programming language of MySQL
- Used for database programming
  E.g. HRA_calc, TAX_calc, Attendance_calc, etc.
- Used for server-side data processing



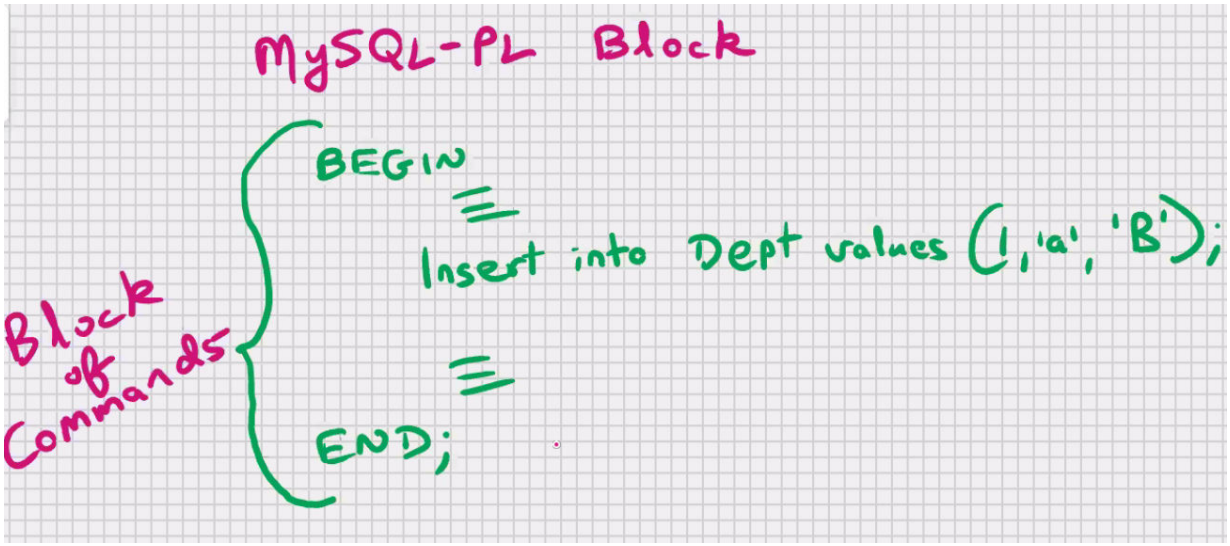
- MySQL -PL program can be called through MySQL command line client, MySQL workbench, oracle forms, oracle reports, oracle menus, oracle graphics, oracle apex(application express),java, ms .net, etc.
- Can be called trough any front-end s/w
- Few 4GL features

Mysql> call hra_calc();

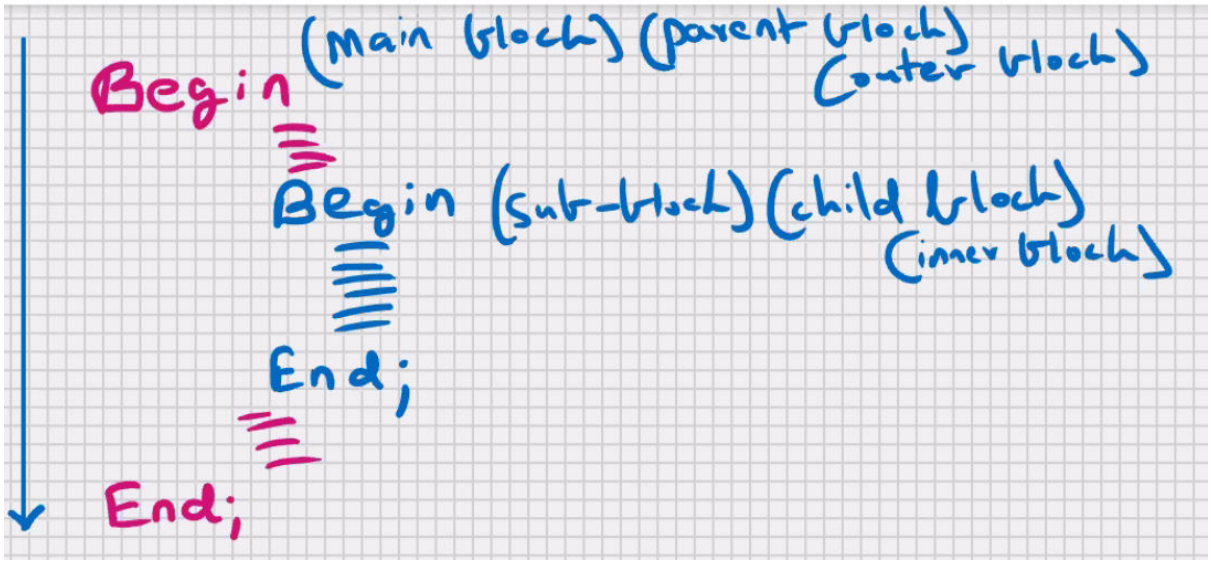
Begin

Insert into dept values(1,’a’,’B’);

End;



- Known as block level language
- Benefits of block within block



1. Modularity
2. Control scope of variables (form of data hiding)
    a. (form of Encapsula)
3. Efficient error management using exceptions

- Screen input and screen output is not allowed (scanf, printf not available)
- Used only for processing
- You can use select statement inside the block but its not recommended
- SQL commands that are allowed inside MySQL-PL:- DDL,DML,DQL,DTL/TCL
- DCL commands are not inside MySQL-PL program

To store output of MySQL-PL program:-

```
Create table tempp
(
Fir int,
Sec char (15)
);
```

- MySQL-PL programs are written in the form of stored procedures

STORED OBJECTS

- Objects that are stored in the database
- E.g. create … tables, indexes, views
- Anything that you do with create command is a stored object

STORED PROCEDURES

- Routine (set of commands) that has to be called explicitly
- Global procedures
- Can be call MySQL command line client, MySQL workbench, java, MS.net etc.
- Can be called through any frontend s/w
- Stored in the database in the compiled format
- Execution will be very fast

- Hiding source code from end-user
- Execution takes place in server RAM
- Procedure can have local variables
- Within the procedure, you can have IF statement, loops, cursors, etc.
- One procedure can call another procedure (concept of calling procedure and called procedure)
- Procedure can call itself (known as recursion)
- You can pass parameters to a procedure (to make it flexible)
- Overloading of stored procedures is not allowed (you cannot create 2 or more procedures with the same name, even if the number of parameters passed is different, or the datatype of parameters passed is different)

Call abc ();

MySQL -> MySQL-PL
Oracle -> PL/SQL (procedural Language SQL)
MS SQL server -> T-SQL (Transact SQL)

#1

```
Delimiter //
create procedure abc ()
begin
        insert into tempp values (1,'Hello');
        commit;
end; //
delimiter;

call abc ();
select * from tempp;
truncate table tempp;
```

→> read, compile, plan and store in the DB in the compiled format
Procedure created.

Drop procedure abc;

- ; is known as terminator (denotes end of command)
- ; is also known as delimiter

#2

```
Delimiter //
Create procedure abc ()
Begin
        Declare x int;
        Set x = 10;
        Insert into tempp values (x, 'Hello');
End; //
Delimiter;

Call abc ();
Select * from tempp;
```

- In MySQL if you declare the variable and you don't initialize, then by **default it will store a null value**
- You can declare a variable and initialize it simultaneously

#3

```
Delimiter //
Create procedure abc ()
Begin
        Declare x int default 10;
        Insert into tempp values (x, 'Hello');
End; //
Delimiter;

Call abc ();
Select * from tempp;
```

#4

```
Delimiter //
Create procedure abc ()
Begin
        Declare x char (15) default ‘CDAC’;
        Insert into tempp values (1, x);
End; //
Delimiter;

Call abc ();
Select * from tempp;
```

For char, varchar, date, time, datetime use ‘ ’ (single quotes)

#5

```
Delimiter //
Create procedure abc ()
Begin
        Declare x char(15) default ‘KING’;
        Declare y float default 3000;
        Declare z float default 0.4;
        Declare hra float;
        Set hra = y*z;
        Insert into tempp values (y, x);
        Insert into tempp values (hra, ‘HRA’);
End; //
Delimiter;

Call abc ();
Select * from tempp;
```

- In MySQL, float to int -> implicit conversion (auto-rounding takes place)

#6

```
Delimiter //
Create procedure abc (x char (15), y float, z float)
Begin
        Declare hra float;
        Set hra = y*z;
        Insert into tempp values (y, x);
        Insert into tempp values (hra, 'HRA');
End; //
Delimiter;

Call abc('KING',3000,0.4);
Call abc('SCOTT',2500,0.3);

Select * from tempp;
```

#7

```
Delimiter //
Create procedure abc (x char (15), y float, z float)
Begin
        --Single line comment
        /* Multi line
                comment*/
End; //
Delimiter;
```

Comments -> internal documentation

- 1 comment min every 2 statements

Drop procedure abc;

To see which all procedures you have created:-

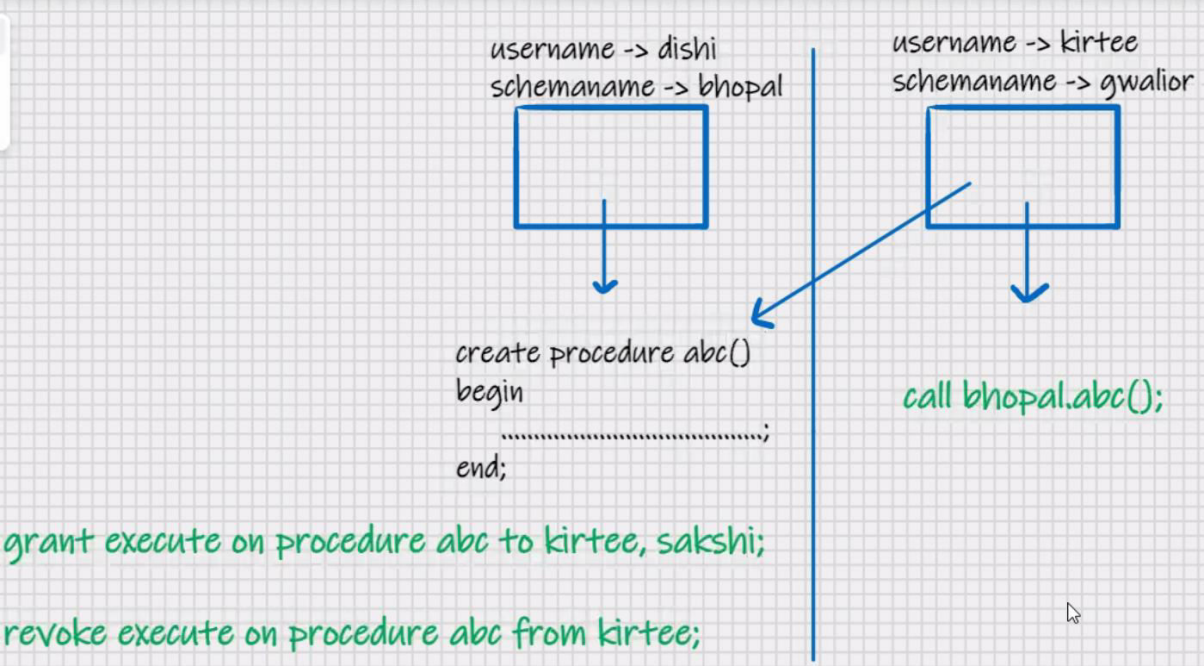
Show procedure status; <- show all procedure in all schemas

Show procedure status where db = 'bhopal';

Show procedure status where name like ‘a%’;

To view the source code of stored procedure:-

Show create procedure abc;



EMP TABLE

ENAME	SAL	JOB	DEPTNO
SCOTT	3000	CLERK	10
KING	5000	MANAGER	20

TEMPP TABLE

FIR	SEC

#1

```
delimiter //
create procedure abc()
begin
      declare x int ;
      select sal into x from emp
      where ename = 'KING' ;
      /* processing , e.g. set hra = x*0.4, etc */
      insert into tempp values (x, 'KING') ;
end; //
delimiter ;
```

Select columnname into variablename from …….
Where …… ;

- X variable should have same datatype and same width as EMP table SAL column

#2

```
delimiter //
create procedure abc(y char (15))
begin
```

```
        declare x int ;
        select sal into x from emp
        where ename = y;
        /* processing , e.g. set hra = x*0.4, etc */
        insert into tempp values (x, y) ;
end; //
delimiter ;

call abc ('KING') ;
```

#3

```
delimiter //
create procedure abc()
begin
        declare x int ;
        declare y char(15) ;
        select sal, job into x, y from emp
        where ename = 'KING';
        /* processing , e.g. set hra = x*0.4, set y = lower(y) etc */
        insert into tempp values (x, y) ;
end; //
delimiter ;

call abc () ;
```

Decision making using IF statement

EMP

ENAME	SAL
KING	5000

#4

```
Delimiter //
Create procedure abc()
Begin
        Declare x int;
        Select sal into x from emp where ename = 'KING' ;
        If x > 4000 then
                Insert into tempp values (x, 'High sal') ;
        End if;
End ; //
Delimiter ;
```

```
If <condition> then
        …… ;
        …… ;
End if ;
```

==========

EMP

ENAME	SAL
KING	3000

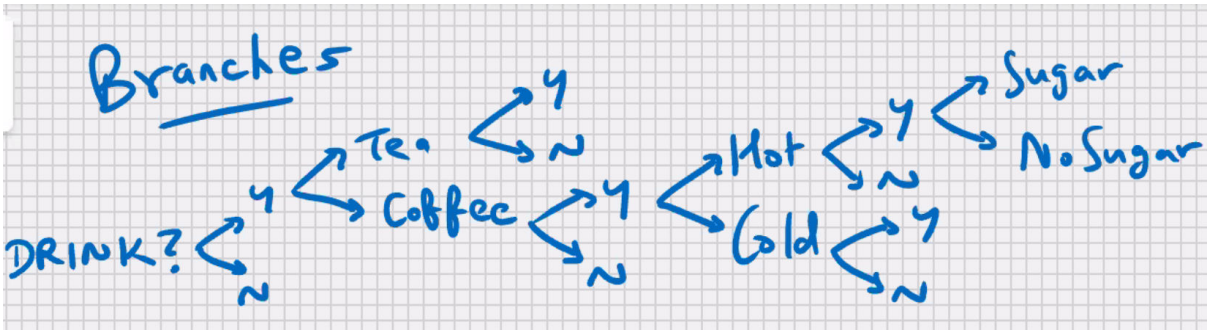
#5

```
Delimiter //
Create procedure abc()
Begin
        Declare x int;
        Select sal into x from emp where ename = 'KING' ;
        If x > 4000 then
                Insert into tempp values (x, 'High sal') ;
        Else
                Insert into tempp values (x, 'Low sal') ;
        End if;
```

```
End ; //
Delimiter ;
```

#6

```
Delimiter //
Create procedure abc()
Begin
        Declare x int;
        Select sal into x from emp where ename = 'KING' ;
        If x > 4000 then
                Insert into tempp values (x, 'High sal') ;
        Else
                If x < 4000 then
                        Insert into tempp values (x, 'Low sal') ;
                Else
                        Insert into tempp values (x, 'Medium sal') ;
                End if;

        End if;
End ; //
Delimiter ;
```



#7

```
Delimiter //
Create procedure abc()
Begin
        Declare x int;
        Select sal into x from emp where ename = 'KING' ;
        If x > 4000 then
                Insert into tempp values (x, 'High sal') ;
        Elseif x < 4000 then
                        Insert into tempp values (x, 'Low sal') ;
        Else
                        Insert into tempp values (x, 'Medium sal') ;
        End if;
End ; //
Delimiter ;
```

```
If x>4000 and x<5000 then
        …….. ;
        …….. ;
ElseIf x between 3000 and 4000 then
        …….. ;
        …….. ;
ElseIf round(x) < 3000 then
        …….. ;
        …….. ;
ElseIf <condition> then
        …….. ;
        …….. ;
End if;
```

#8

```
Delimiter //
Create procedure abc()
Begin
        Declare x Boolean default TRUE;
        If x then
                Insert into tempp values (1, 'Mumbai') ;
        End if;
End ; //
Delimiter ;
```

#9

```
Delimiter //
Create procedure abc()
Begin
        Declare x Boolean default FALSE;
        If not x then
                Insert into tempp values (1, 'Mumbai') ;
        End if;
End ; //
Delimiter ;
```

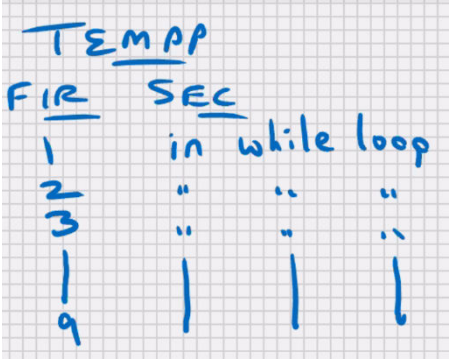
MySQL-PL LOOPS

- For repetitive /iterative processing

While loop

- Check for some condition before entering the loop

```
While expression DO
        …….. ;
        …….. ;
End while ;
```



#10

```
Delimiter //
Create procedure abc ()
Begin
        Declare x int default 1;
        While x < 10 do
                Insert into tempp values (x, 'in while loop');
                Set x = x+1;
        End while;
End ; //
Delimiter ;
```

#11

```
Delimiter //
Create procedure abc ()
Begin
        Declare x int default 1;
        While x < 10 do
                Insert into tempp values (x, 'in while loop');
                /*Set x = x+1;*/  infinite loop
        End while;
End ; //
Delimiter ;
```

#12

```
Delimiter //
Create procedure abc ()
Begin
        Declare x int default 1;
        Declare y int default 1;
        While x < 10 do
                While y < 10 do
                        Insert into tempp values (y, 'in y loop');
```

```
                Set y = y+1;
            End while;
            Insert into tempp values (x, 'in x loop');
            Set x = x+1;
        End while;
End ; //
Delimiter ;
```

#13

```
Delimiter //
Create procedure abc ()
Begin
        Declare x int default 1;
        Declare y int default 1;
        While x < 10 do
            While y < x do
                Insert into tempp values (y, 'in y loop');
                Set y = y+1;
            End while;
            Insert into tempp values (x, 'in x loop');
            Set x = x+1;
        End while;
End ; //
Delimiter ;
```

Repeat Loop

- Similar to Do While loop
- There's no coditon to enter the loop, but there is a condition to exit the loop
- It will execute at least once
- e.g., Outer join

repeat

…….. ;

…….. ;

Until expression

End repeat;

#14

```
Delimiter //
Create procedure abc ()
Begin
        Declare x int default 1;
        repeat
                Insert into tempp values (x, 'in loop');
                Set x = x+1;
        Until x > 5
        End repeat;
End ; //
Delimiter ;
```

Loop, Leave, and Iterate statement: -

- **Leave** statement allows you to exit the loop (similar to **break** statement of c programming)
- **Iterate** statement allows you skip the entire code under it and start a new iteration (similar to '**continue**' statement of C programming)
- Loop statement allows you to execute a block of code repeatedly with an additional flexibility of using a loop label (you can give a name to the loop)

#15

```
Delimiter //
Create procedure abc ()
Begin
        Declare x int default 1;
        Pqr_loop: loop
                If x > 10 then
                                Leave pqr_loop;
                End if;
                Set x = x+1;
                If mod(x,2) != 0 then
                                Iterate pqr_loop;
                Else
                        Insert into tempp values (x, 'loop') ;
                End if;
        End loop;
End ; //
Delimiter ;
```

Global variables (session variables):-

```
Set @x = 10;

Select @x from dual;     -> 10

Set @x = @x + 1;
Select @x from dual;      -> 11
```

CURSORS (MOST IMPORTANT TOPIC)

EMP TABLE

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

TEMPP TABLE

FIR	SEC

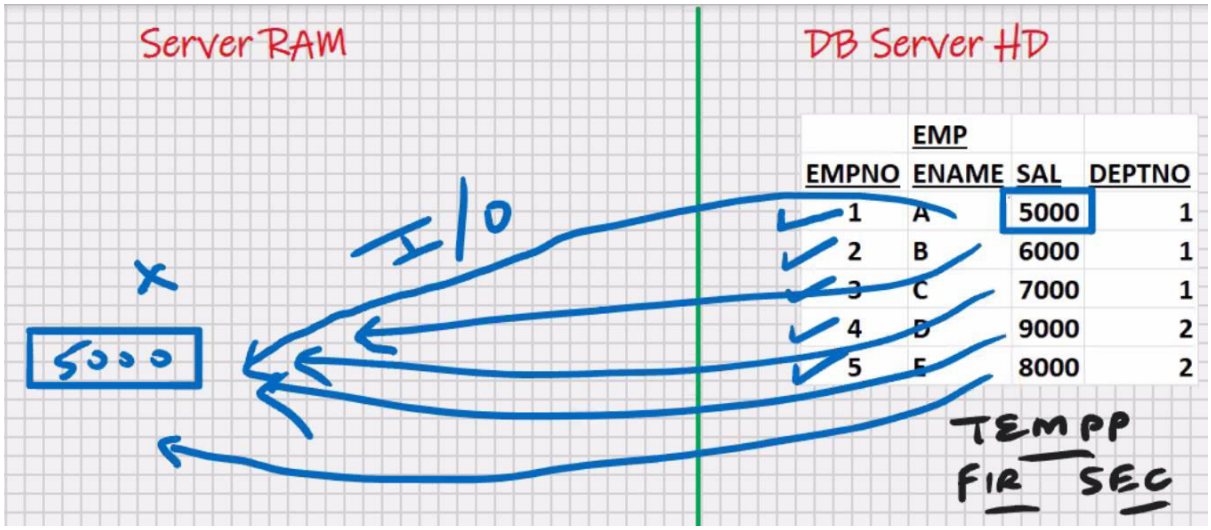
- Present in all RDBMS, some DBMS and some of the programming languages and some front-ends also

```
Create table emp (
Empno int,
Ename varchar (15),
Sal int,
Deptno int
);
```

- Cursor is a type of a variable

```
Declare x int;
Declare hra flaot;
Select sal into x from emp where empno = 1;
Set hra = x*0.4;
Insert into tempp values (hra, 'HRA');
Select sal into x from emp where empno = 2;
Set hra = x*0.4;
Insert into tempp values (hra, 'HRA');
```

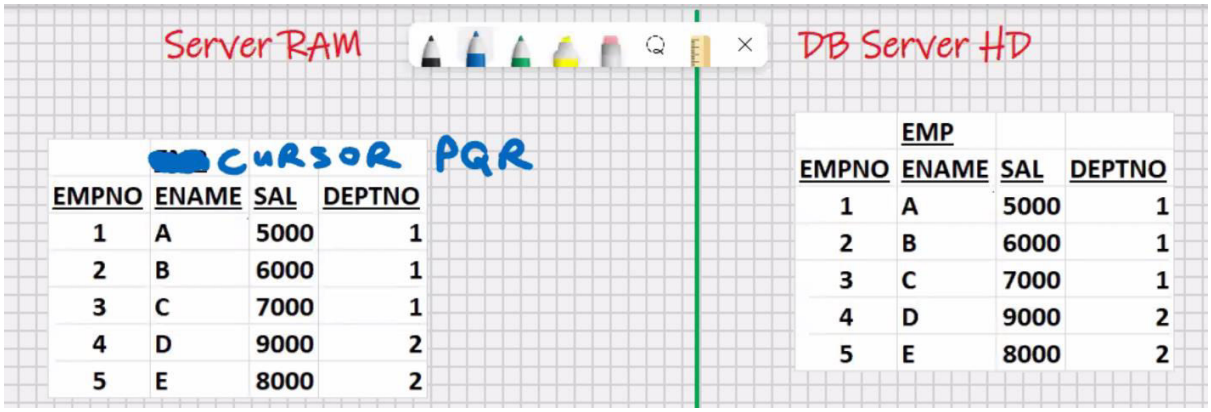
Select sal into x from emp where empno = 3;
Set hra = x*0.4;
Insert into tempp values (hra, ‘HRA’);
Select sal into x from emp where empno = 4;
Set hra = x*0.4;
Insert into tempp values (hra, ‘HRA’);
Select sal into x from emp where empno = 5;
Set hra = x*0.4;
Insert into tempp values (hra, ‘HRA’);



This is slow process

Declare y int default 1;
Declare x int;
Declare hra flaot;
While y < 6
    Select sal into x from emp where empno = y;
    Set hra = x*0.4;
    Insert into tempp values (hra, ‘HRA’);
    Set y = y+1
End while;

Solution using Cursors



EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

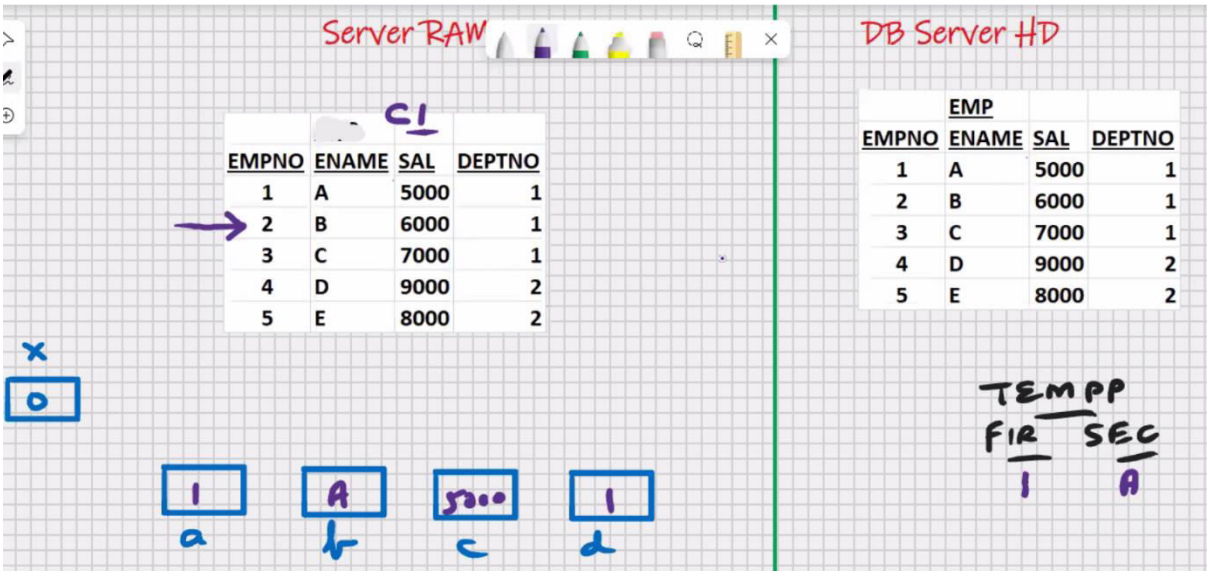
EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

- Used for storing **multiple rows**
- Used for processing storing multiple rows
- Used for handling multiple rows
- Used for storing the **data temporarily**
- Cursor is similar to a **2D array**
- Cursor is based on select statement

Declare pqr cursor for select * from emp;
Declare pqr cursor for select ename, sal from emp;
Declare pqr cursor for select ename, sal from emp where deptno=1;
Declare pqr cursor for select ename, sal from emp where deptno=1
Order by 1;

#16

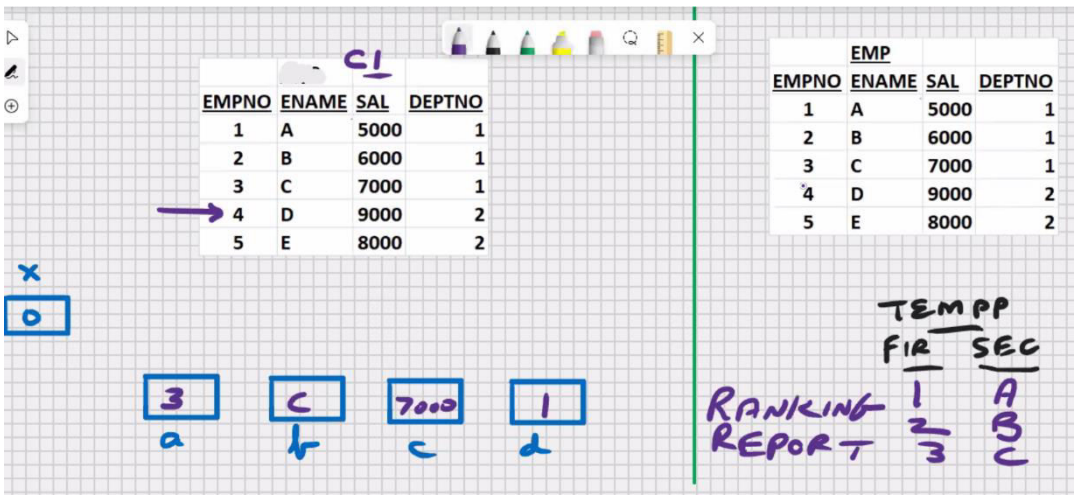
```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar(15);
        Declare c int;
        Declare d int;
        Declare x int default 0;
        Declare c1 cursor for select * from emp; <- cursor
        declaration/definition (at this point, the cursor does not contain
        any data)
        Open c1;  <- this will open the cursor c1, executes the select
        statement , and it will populate the cursor c1
        While x < 5 do
                Fetch c1 into a, b, c, d;   <-Fetches the next row
                /*processing, e.g. set hra = c*0.4, etc */
                Insert into tempp values (a, b);
                Set x = x+1;
        End while;
        Close c1;    <-this will close the cursor c1 and free the RAM
End ; //
Delimiter ;
```



- Cursor is read only variable
- The data that is present inside the cursor, it cannot be manipulated
- You will have to fetch 1 row at a time into some intermediate ‘a’,’b’,’c’,’d’ variables, and do your processing with those variables
- You can only fetch sequentially (top to bottom)
- In MySQL you cannot fetch backwards in a oracle
- You can only fetch 1 row at a time

#17

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar(15);
        Declare c int;
        Declare d int;
        Declare x int default 0;
        Declare c1 cursor for select * from emp;
        Open c1;
        While x < 3 do
                Fetch c1 into a, b, c, d;
                Insert into tempp values (a, b);
                Set x = x+1;
        End while;
        Close c1;
End ; //
Delimiter ;
```



#18

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar(15);
        Declare c int;
        Declare d int;
        Declare x int default 0;
        Declare c1 cursor for select * from emp;
        Open c1;
        While x < 10 do
                        Fetch c1 into a, b, c, d;     -> ERROR when x=5
                        Insert into tempp values (a, b);
                        Set x = x+1;
        End while;
        Close c1;
End ; //
Delimiter ;
```

#19

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar(15);
        Declare c int;
        Declare d int;
        Declare x int default 0;
        Declare y int;
        Declare c1 cursor for select * from emp order by sal desc;
        Select count (*) into y from emp;
        Open c1;
        While x < y do
                        Fetch c1 into a, b, c, d;
                        Insert into tempp values (a, b);
                        Set x = x+1;
        End while;
        Close c1;
End; //
Delimiter;
```

- Declare a continue handler for not found event (exception)
- Not found is a cursor attribute, it returns a Boolean true value if the last fetch was unsuccessful, and false value if the last fetch was successful

#20

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar(15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select * from emp;
        Declare continue handler for not found set y = 1;
        Open c1;
        Cursor_c1_loop: loop
                        Fetch c1 into a, b, c, d;
                        If y = 1 then
                                        Leave cursor_c1_loop;
                        End if;
                        Insert into tempp values (a, b);
        End loop cursor_c1_loop;
        /* Close c1; */      <- closing is optional if it is last line
End; //
Delimiter;
```

- You cannot reset the cursor pointer

```
Open c1;
Open c1;
```

- You cannot same twice, you will get an error that cursor is already open, you will have to close the cursor before you reopen it

To reset the cursor pointer: -

```
Close c1;
Open c1;
```

#21

```
Delimiter //
Create procedure abc ()
Begin
        Declare a varchar(15);
        Declare b int;
        Declare y int default 0;
        Declare c1 cursor for select ename, sal from emp;
        Declare continue handler for not found set y = 1;
        Open c1;
        Cursor_c1_loop: loop
                        Fetch c1 into a, b;
                        If y = 1 then
                                        Leave cursor_c1_loop;
                        End if;
                        Insert into tempp values (b, a);
        End loop cursor_c1_loop;
        Close c1;
End; //
Delimiter;
```

#22

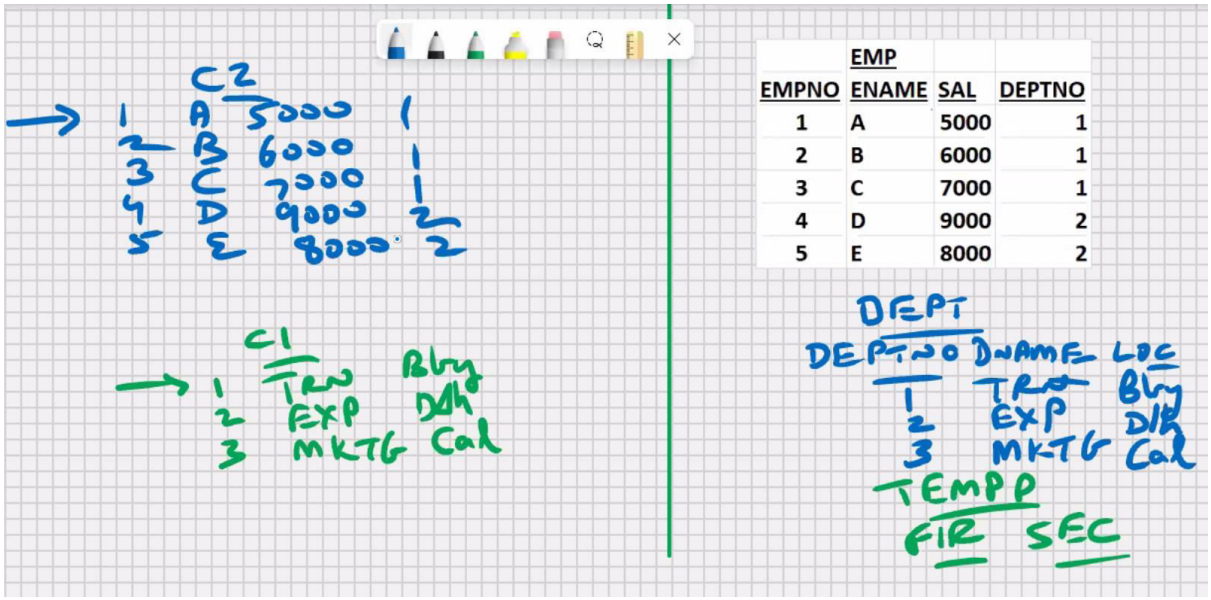
```
Delimiter //
Create procedure abc ()
Begin
        Declare a int ;
        Declare b varchar(15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select * from emp where deptno = 1;
        Declare continue handler for not found set y = 1;
        Open c1;
        Cursor_c1_loop: loop
                        Fetch c1 into a, b, c, d;
                        If y = 1 then
                                        Leave cursor_c1_loop;
                        End if;
                        Insert into tempp values (c, b);
        End loop cursor_c1_loop;
        Close c1;
End; //
Delimiter;
```

#23

```
Delimiter //
Create procedure abc (dd int)
Begin
        Declare a int ;
        Declare b varchar(15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select * from emp where deptno = dd;
        Declare continue handler for not found set y = 1;
        Open c1;
        Cursor_c1_loop: loop
                        Fetch c1 into a, b, c, d;
                        If y = 1 then
                                        Leave cursor_c1_loop;
                        End if;
                        Insert into tempp values (c, b);
        End loop cursor_c1_loop;
        Close c1;
End; //
Delimiter;

Call abc (1);
Call abc (2);
```

DEPT TABLE

DEPTNO	DNAME	LOC
1	TRN	MUMBAI
2	EXP	DELHI
3	MRKG	CALCUTTA



#24

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int ;
        Declare b varchar(15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select * from dept;
        Declare c2 cursor for select * from emp;
        Declare continue handler for not found set y = 1;
        …… ;
        Open c1;
        Open c2;
        …… ;
        Close c1;
        Close c2;
End; //
Delimiter;
```

- No upper limit on the number of cursors that you can open at a time
- The only restriction would be size of server RAM (it should be large enough to manage so much data)

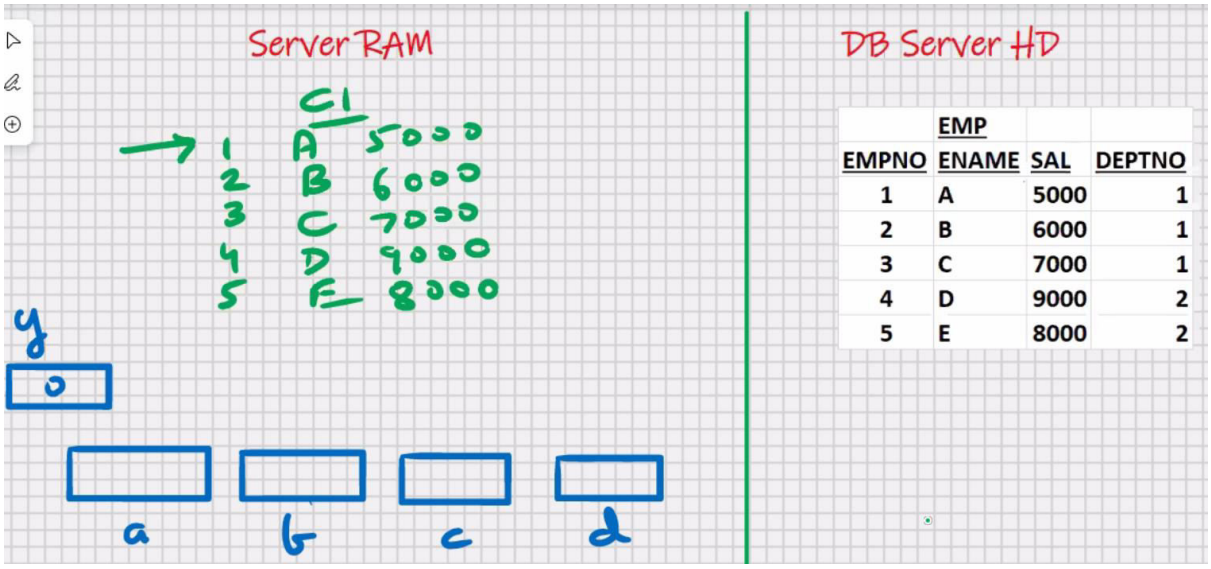
#25

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar (15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select empno, dname from dept emp,
        dept where dept.deptno = emp.deptno ;
        …… ;
        Open c1;
        …… ;
        Close c1;
End; //
Delimiter;
```

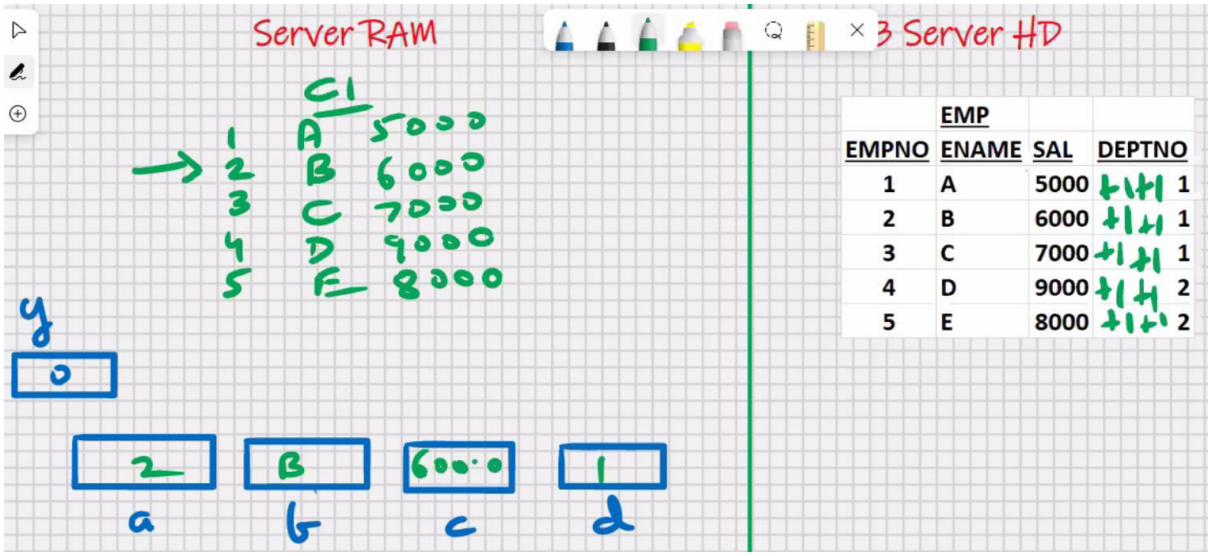
MySQL-PL - Cursors

#1

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar (15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select * from emp;
        Declare continue handler for not found set y = 1;

        Open c1;
        Cursor_c1_loop: loop
                Fetch c1 into a, b, c, d;
                If y =1 then
                        Leave cursor_c1_loop;
                End if;
                Update emp set sal = sal + 1;
        End loop cursor_c1_loop;
        Close c1;
End; //
Delimiter;
```



	EMP		
EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2



	EMP		
EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

#2

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar (15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select * from emp;
        Declare continue handler for not found set y = 1;

        Open c1;
        Cursor_c1_loop: loop
                Fetch c1 into a, b, c, d;
                If y =1 then
                        Leave cursor_c1_loop;
                End if;
                If c>7000 then
                Update emp set sal = sal + 1;
                End if;
        End loop cursor_c1_loop;
        Close c1;
End; //
Delimiter;
```

#3

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar (15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select * from emp for update;
        Declare continue handler for not found set y = 1;

        Open c1;
        Cursor_c1_loop: loop
                Fetch c1 into a, b, c, d;
                If y =1 then
                        Leave cursor_c1_loop;
                End if;
                If c>7000 then
                Update emp set sal = sal + 1 where empno = a;
                End if;
        End loop cursor_c1_loop;
        Close c1;
        Commit;
End; //
Delimiter;
```

#4

```
Delimiter //
Create procedure abc ()
Begin
        Declare a int;
        Declare b varchar (15);
        Declare c int;
        Declare d int;
        Declare y int default 0;
        Declare c1 cursor for select * from emp for update;
        Declare continue handler for not found set y = 1;

        Open c1;
        Cursor_c1_loop: loop
                Fetch c1 into a, b, c, d;
                If y =1 then
                        Leave cursor_c1_loop;
                End if;
                If c>7000 then
                        delete emp set sal = sal + 1 where empno = a;
                End if;
        End loop cursor_c1_loop;
        Close c1;
        Commit;
End; //
Delimiter;
```

Cursors uses

- Storing/processing multiple rows
- **USED TO LOCK THE ROWS MANUALLY**

#5

```
Delimiter //
Create procedure abc()
begin
        Declare c1 cursor for Select * from emp for update;
        Open c1;
        Close c1;
End; //
Delimiter ;
Call abc();
```

- **LOCKS ARE AUTOMATICALLY RELEASED WHEN YOU ROLLBACK OR COMMIT**

#6

```
Delimiter //
Create procedure abc ()
begin
        Declare c1 cursor for Select * from emp
        Where deptno = 1 for update;
        Open c1;
        Close c1;
End; //
Delimiter ;
Call abc();
```

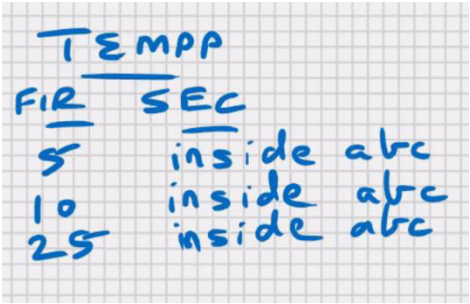
Parameters are of 3 types

1. **In (by default)**

- Read only
- Can pass a constant, variable and expression
- Call by value
- Fastest in terms of processing speed
- If you don't want to return a value, then use IN parameter

#7

```
Delimiter //
Create procedure abc (in y int)
Begin
        --Set y = 100;   <- ERROR
        Set x = y + 10 ; <- Allowed
        Insert into tempp values (y, 'inside abc');
End ; //
Delimiter ;
Call abc (5);

Set @x = 10;
Call abc(@x);

Set @x = 10;
Call abc(2*@x+5);
```



#8

```
Delimiter //
Create procedure abc (in y int)
Begin
        Insert into tempp values (y, 'inside abc');
End ; //
Delimiter ;
```

```
Delimiter //
Create procedure pqr ()
Begin
        Declare x int default 10;
        Call abc(5) ;
        Call abc(x);
        Call abc(2*x+5);
End ; //
Delimiter ;
Call pqr();
```

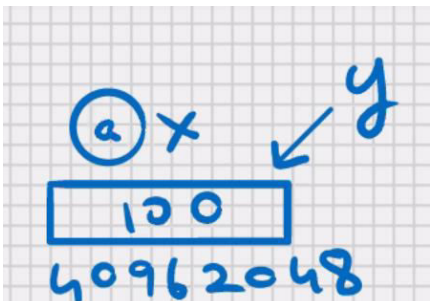
==========

2. **Out**

- Write only
- Can pass variables only
- Call by reference
- Procedure can return a value indirectly if you call by reference
- **MOST SECURE**
- if you are working on a public network, e.g. Internet (e.g. username, password, OTP, etc.)

#9

```
Delimiter //
Create procedure abc (out y int)
Begin
        -- Set x = y ; -> ERROR      cant read from y
        Set y = 100;

End ; //
Delimiter ;

Set @x = 10 ;
```



Select @x from dual;

10

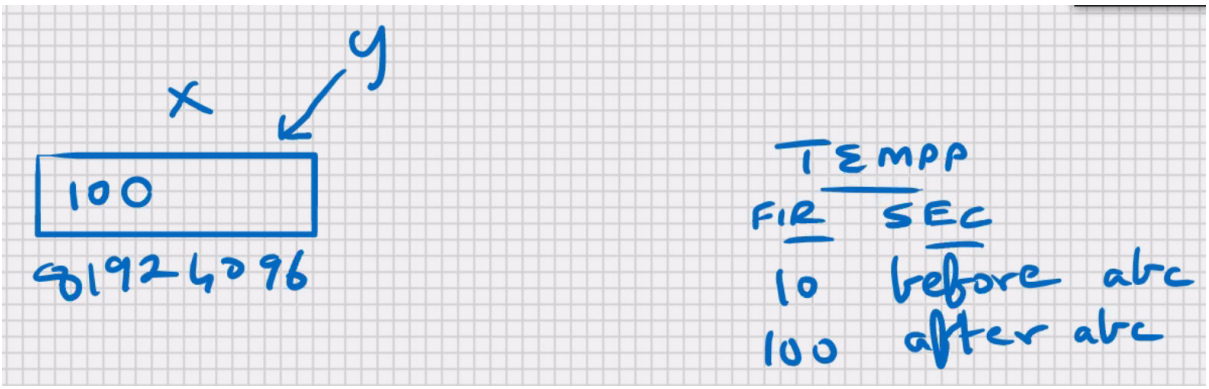
Call abc (@x);

Select @x from dual;

100

#10

```
Delimiter //
Create procedure abc (out y int)
Begin
        set y = 100;
End ; //
Delimiter ;

Create procedure pqr ()
Begin
        Declare x int default 10;
        Insert into tempp values (x, 'before abc');
        Call abc(x);
        Insert into tempp values(x, 'after abc');
End ; //
Delimiter ;
Call pqr();
```



==========

3. **inout**

- Read and Write
- Can pass variables only
- Call by reference
- Procedure can return a value indirectly if you call by reference
- **MOST POWERFUL**
- if you are working on a local network

#11

```
Delimiter //
Create procedure abc (inout y int)
Begin
        Set y = y * y * y;
End ; //
Delimiter ;

Set @x = 10 ;
Select @x from dual;
        10

Call abc (@x);
Select @x from dual;
        100
```

#12

```
Delimiter //
Create procedure abc (out y int)
Begin
        set y = y*y*y;
End ; //
Delimiter ;
```

```
Create procedure pqr ()
Begin
        Declare x int default 10;
        Insert into tempp values (x, 'before abc');
        Call abc(x);
        Insert into tempp values(x, 'after abc');
End ; //
Delimiter ;
Call pqr();
```

Stored objects

- objects that are stored in the database
- e.g. create ……. Tables, indexes, views, procedures
- anything that you do create command is a stored object

STORED FUNCTIONS

- Routine that **returns a value** directly and compulsorily
- Global functions
- Can be called through MySQL command line client, MySQL Workbench, MySQL- PL, Java, MS.NET ,etc
- Can be called through any front-end s/w;
- Unlike a procedure, a function cannot be called by itself, because a function returns a value, and that value has to be stored somewhere
- Therefore a function has to be equated with a variable, or it has to be a part of some expression
- Etc. points and benefits are same as procedures.
- In parameter only

Call y = abc();

Functions are of 2 types:

1.Deterministic

2.Not Deterministic

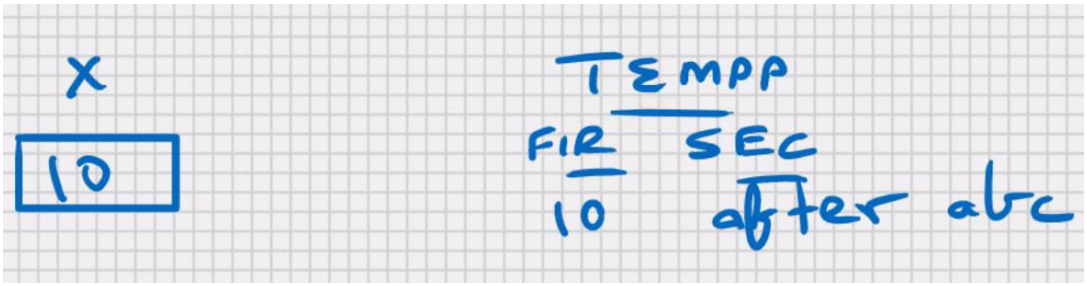
- for the same input parameters, if the stored function returns the same result, it is considered deterministic, and otherwise the stored function is not deterministic
- you have to decide whether a stored function is deterministic or not
- if you declare it incorrectly, the stored function may produce an unexpected result, or the available optimization is not used which degrades the performance
- only the current date and time are used inside the body of the function, then it will be not deterministic, else mostly it is going to be deterministic

#13

```
delimiter //
create function abc ()
returns int
deterministic
begin
        return 10;
end; //
delimiter ;
```

Function created.

```
delimiter //
create procedure pqr ()
begin
        declare x int;
        set x = abc ();   -- 10
        insert into tempp values (x, 'after abc');
end; //
delimiter ;

call pqr();
```



#14

```
delimiter //
create function abc (y int)
returns int
deterministic
begin
        return y*y;
end; //
delimiter ;


delimiter //
create procedure pqr ()
begin
        declare x int;
        set x = abc (10);
        insert into tempp values (x, 'after abc');
end; //
delimiter ;

call pqr ();
```

difference between stored function and stored procudures

- stored function can be called in select statement

```
select abc(sal) from emp;
```

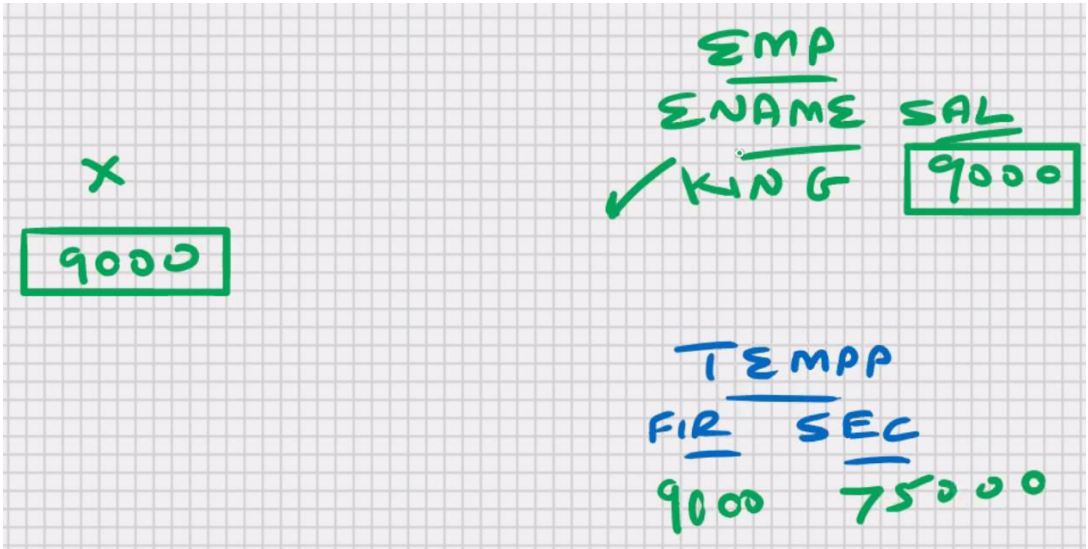
- stored function can be called in sql commands

```
select abc (10) from dual;
delete from emp where abc(sal) = 1000000;
```

#15

```
delimiter //
create function abc (y int)
returns boolean
deterministic
begin
        if y > 5000 then
        return TRUE;
        else
        return FALSE;
        end if;
end; //
delimiter ;
```

```
delimiter //
create procedure pqr ()
begin
        declare x int;
        select sal into x from emp where ename = 'KING';
        if abc(x) then
        insert into tempp values (x, '> 5000');
        else
        insert into tempp values (x, '<= 5000');
        end if;
end; //
delimiter ;

call pqr ();
```



to drop the function: -
drop function abc;

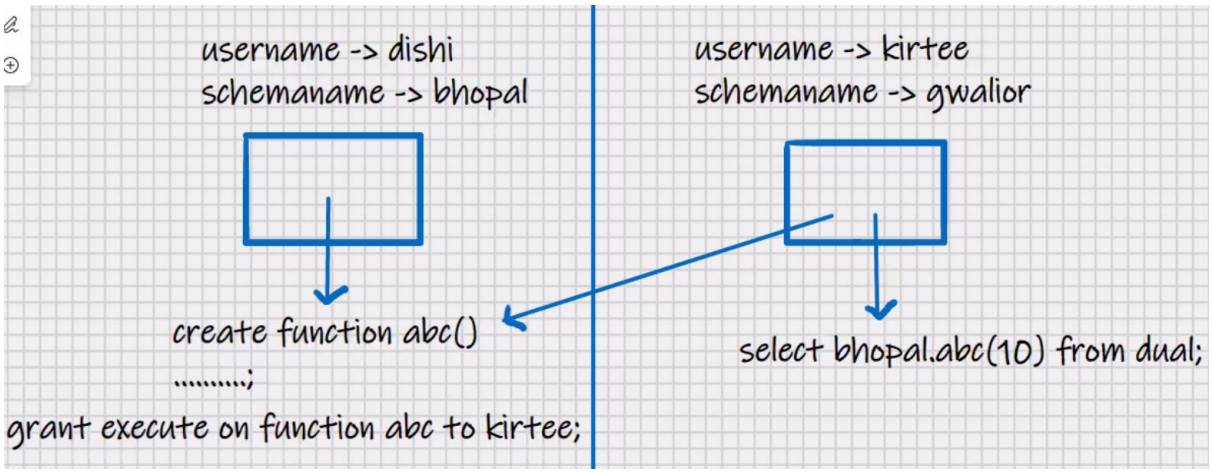
to see which all functions are created: -
show function status; -> shows all functions in all schemas

show function status where db = 'cdac';
show function status where name like 'a%';

to view the source code of stored function: -
show create function abc;

to share the function with n other users: -

edac_mysql> grant execute on function abc to scott;
scott_mysql> select cdac.abc() from dual;
edac_mysql> revoke execute on function abc from scott;



MySQL Stored Objects

- Objects that are stored in the databases
- E.g. create tables, indexes, views, procedures, functions

DATABASE TRIGGERS (VERY IMPORTANT)

- Present in some of the RDBMS
- routine (set of commands) that gets executed AUTOMATICALLY when some EVENT takes place
- EVENT -> when something happens
- Triggers are written on tables
- Events are: - (**total 6 events**)

  Before INSERT, After INSERT

  Before DELETE, After DELETE

  Before UPDATE, After UPDATE

EMP

ENAME	SAL	DEPTNO
A	5000	1
B	5000	1
C	5000	1
D	3000	2
E	3000	2

DEPTOT

DEPTNO	SALTOT
1	15000
2	6000

TEMPP

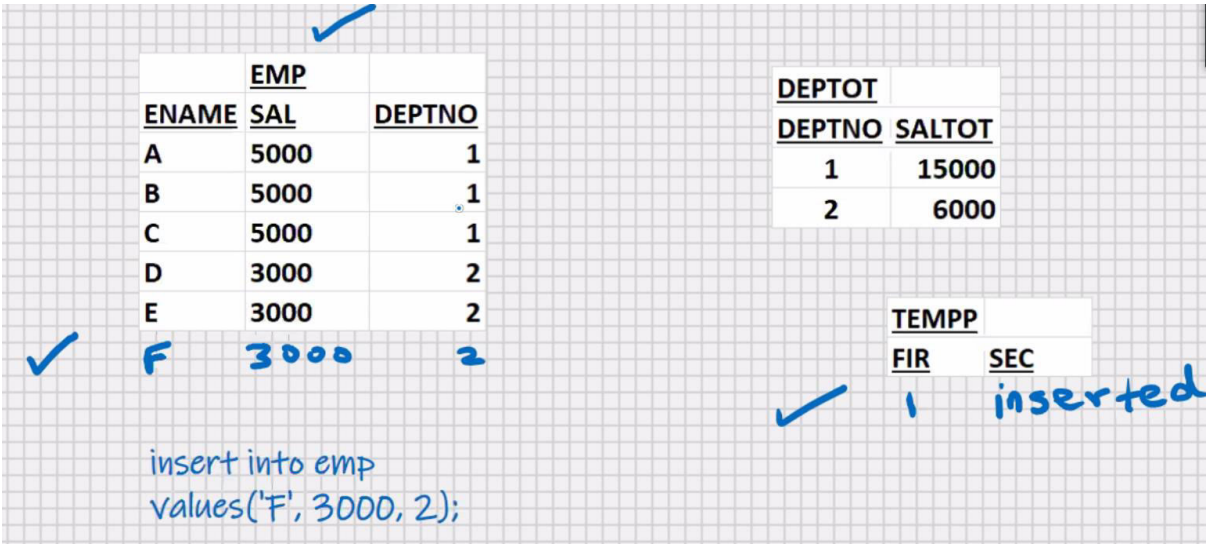
FIR	SEC

select deptno, sum(sal) from emp
group by deptno;

#1

```
delimiter //
create trigger abc
before insert
on emp for each row
begin
        insert into tempp values(1,'inserted') ;
end; //
delimiter ;
```

---> Read, compile, plan, store it in the DB in the compiled format
Trigger created.



#2

```
delimiter //
create trigger abc
after insert
on emp for each row
begin
        insert into tempp values(1,'inserted');
                -- COMMIT; -- rollback and commit are not allowed inside
                  Triggers
end; //
delimiter ;
```

USES: -

- used to maintain logs (AUDIT TRAILS) of insertions
- within the trigger all MySQL-PL statements allowed, e.g. declare variables, declare cursors, IF statement, loops, etc
- MySQL will read, compile, make a plan, and store it in the database in the COMPILED FORMAT
- all triggers are at server level, you may perform your DML operations using any front-end s/w, the triggers will always execute
- within the trigger you can have any processing, full MySQL PL allowed
- If DML operation on table fails, then it will cause the event to fail, and then trigger changes are automatically rolled back
- If trigger fails, then it will cause the event to fail, and then DML operation on table is automatically rolled back
- **YOUR DATA WILL ALWAYS BE CONSISTENT**
- ROLLBACK and COMMIT not allowed inside the trigger
- ROLLBACK or COMMIT is to be specified AFTERWARDS, at the end of transaction
- Whether you COMMIT or ROLLBACK afterwards, the data will always be consistent
- In MySQL all triggers are at ROW LEVEL (they will fire for each row)
- In MySQL you can have max 6 triggers per table
- **After trigger is recommended (this will be more efficient)**

#3

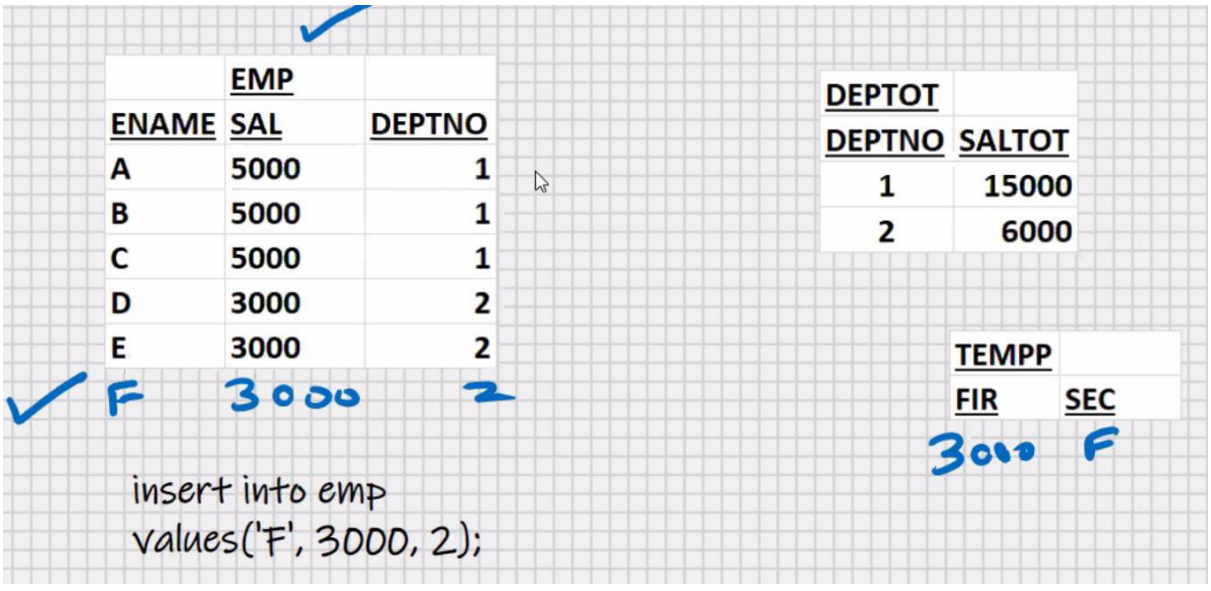
```
delimiter //
create trigger abc
after insert
on emp for each row
begin
        insert into tempp values(1,'inserted',user(), sysdate(), etc.);
end; //
delimiter ;
drop trigger abc;
```

- if you drop the table, then indexes and triggers are dropped automatically (view, procedures remains)

#4

```
delimiter //
create trigger abc
before insert
on emp for each row
begin
        insert into tempp values(new.sal, new.ename);
end; //
delimiter ;
```

- new.ename, new.sal, new.deptno are MySQL created variables



USES: -

- Automatic data duplication, data mirroring, concept of parallel server, concept of standby database in the case of Insert
- Maintain SHADOW tables in the event of insert
- After insert trigger is recommended

#5

```
delimiter //
create trigger abc
before insert
on emp for each row
begin
        set new.ename = upper(new.ename) ;
        insert into tempp values (new.sal, new.ename);
end; //
delimiter ;
```

USES: -

- Data cleansing BEFORE insert
- Data validations BEFORE insert

#6

```
delimiter //
create trigger abc
before insert
on emp for each row
begin
        update deptot set saltot = saltot + new.sal
        where deptno = new.deptno ;
end; //
delimiter ;
```

- Automatic updation of related tables

Show triggers ;
Show all triggers in all schemas

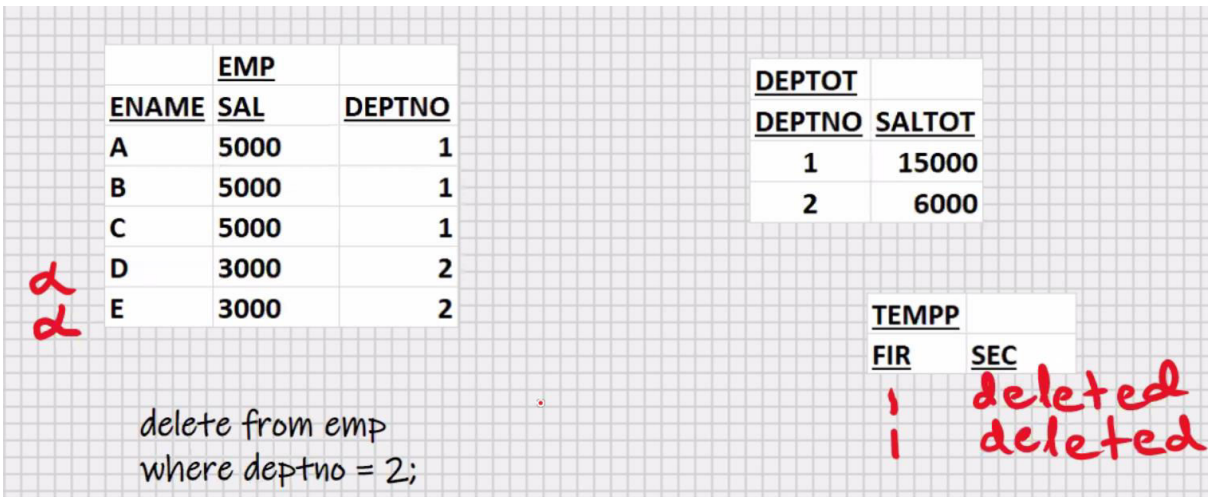
Show trigger from [db_name] ;

Show triggers from Bhopal ;

Select * from information_schema.triggers where trigger_schema = ‘bhopal’

delete from emp where deptno = 2;
#7

```
delimiter //
create trigger pqr
before delete
on emp for each row
begin
        insert into tempp values(1, 'deleted', user(), sysdate());
end; //
delimiter ;
```

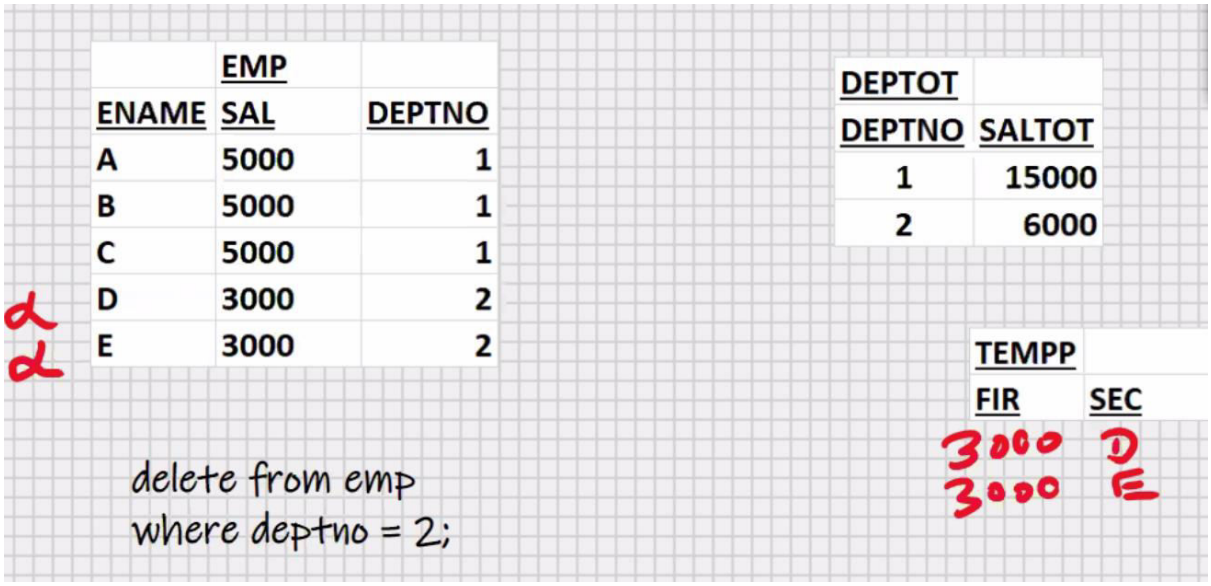


Uses :-

- Maintain logs of deletions
- After delete trigger recommended

#8

```
delimiter //
create trigger pqr
before delete
on emp for each row
begin
        insert into tempp values (old.sal, old.ename);
end; //
delimiter ;
```



- old.name, old.sal, old.deptno are MySQL created variables

Uses:-

- maintain HISTORY tables in the event of delete
- AFTER delete trigger recommended

#9

```
delimiter //
create trigger pqr
before delete
on emp for each row
begin
update deptot set saltot = saltot- old.sal
where deptno = old.deptno;
end; //
delimiter ;
```

Uses:-

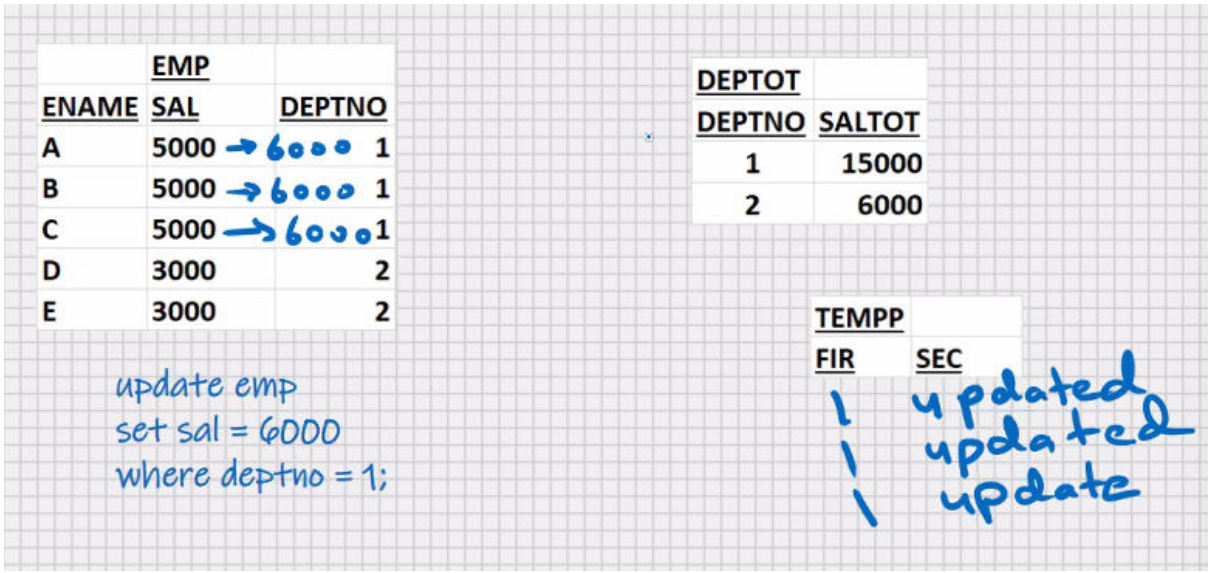
- Automatic updation of related tables
- AFTER delete trigger recommended

Update emp set sal = 6000 where deptno = 1;

#10

```
Delimiter trigger xyz
Before update
On emp for each row
Begin
        Insert into tempp values (1, 'updated');
End ; //
Delimiter ;
```

Trigger created.



Uses:-

- Maintain logs (audit trails) of updations
- AFTER update trigger is recommended

CASCADING TRIGGERS: -

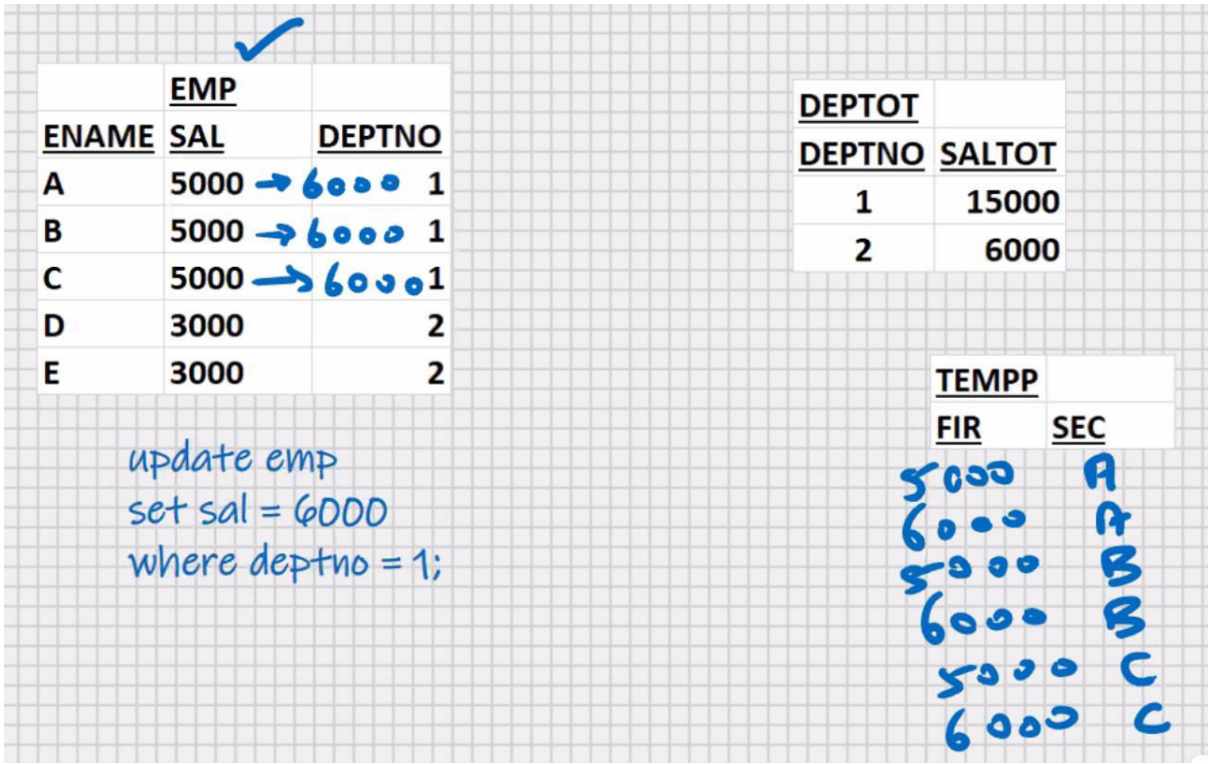
- one trigger causes a second trigger to execute, which in turn causes a third trigger to execute, and so on and so forth
- max upto 32 levels for Cascading triggers
- any kind of power failure, network failure, system failure, PC reboot, window close, improper exit, etc. the entire transaction is automatically rolled back

MUTATING TABLES ERROR -> if some cascading trigger causes one of the previous triggers to execute, then MySQL will not go into infinite loop; you will get an error of Mutating tables and the entire transaction is automatically ROLLED BACK

#11

```
delimiter //
create trigger xyz
before update
on emp for each row
begin
        insert into tempp values (1, 'updated');
end; //
delimiter ;

delimiter //
create trigger xyz2
before insert
on tempp for each row
begin
        delete from deptot .....;
end; //
delimiter ;

delimiter //
create trigger xyz3
```

```
before update
on emp for each row
begin
        insert into tempp values (old.sal, old.ename);
        insert into tempp values (new.sal, new.ename);
end; //
delimiter ;
```



- new.name, new.sal, new.deptno, old.name, old.sal, old.deptno are MySQL created variables
- Maintain SHADOW and HISTORY tables in the event of the update
- Data cleansing BEFORE update
- Data validation BEFORE update

Update emp set sal = 6000 where ename = ‘A’ ;

#12

```
delimiter //
create trigger xyz
before update
on emp for each row
begin
        if old.deptno <> new.deptno or old.sal <> new.sal then
        if old.deptno <> new.deptno then
                update deptot set saltot = saltot – old.sal
                where deptno = old.deptno;
                update deptot set saltot = saltot + new.sal
                where deptno = new.deptno;
        else
                /*if you update only the sal column*/
                update deptot set saltot = saltot - old.sal + new.sal
                where deptno = old.deptno;
        end if;
        end if;
end; //
delimiter ;
```

Uses:-

- Automatic updation of related tables
- AFTER update trigger recommended

aditya ←

	EMP	
ENAME	SAL	DEPTNO
A	5000	1
B	5000	1
C	5000	1
D	3000	2
E	3000	2

```
update emp
set ename = 'aditya'
where ename = 'A';
```

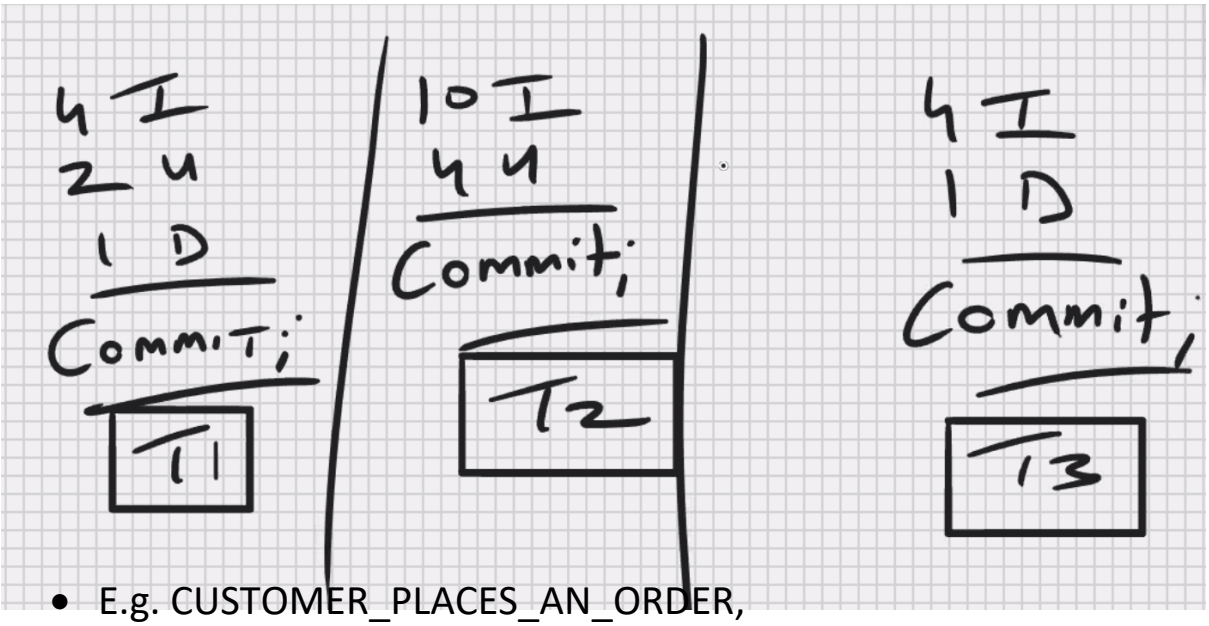
DEPTOT	
DEPTNO	SALTOT
1	15000
2	6000

−5000
+5000

TEMPP	
FIR	SEC

NORMALISATION

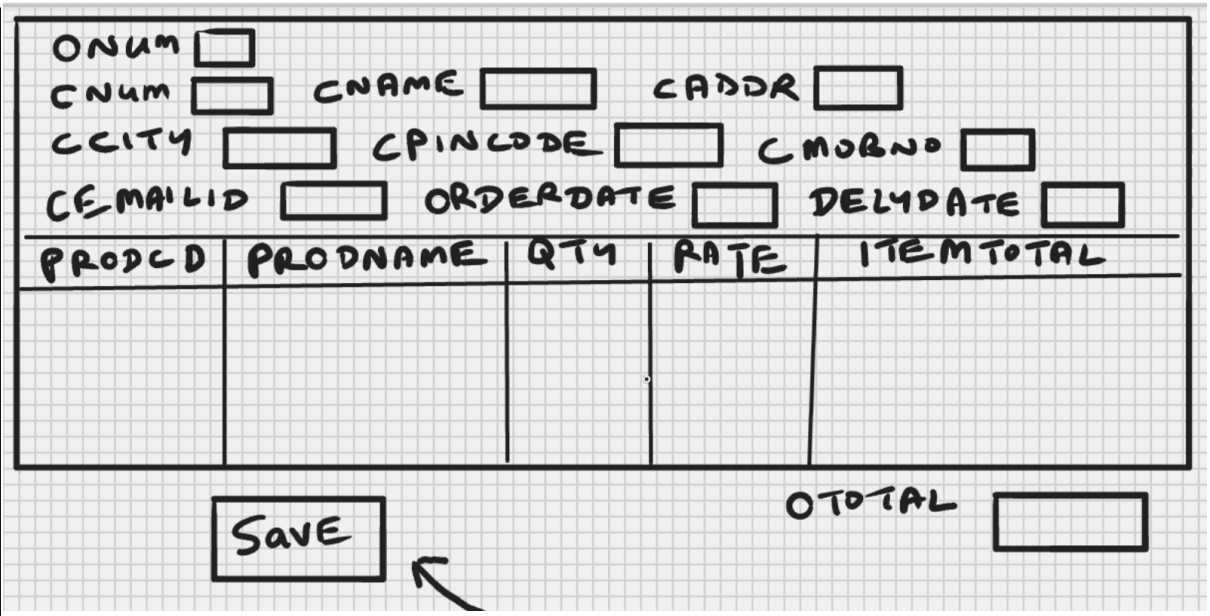
- Applicable to RDBMS, and ORDBMS
- 1^{st} – 4^{th} Normal Form -> RDBMS
- 1^{st} – 9^{th} Normal form -> ORDBMS
- Concept of table design
- What tables to create, structures, columns, datatypes, widths, and constraints
- Based on user requirements
- Part of design phase (1/6)
- Aim of normalisation is to avoid the data redundancy (avoid the unnecessary duplication of data)
- Secondary aim of normalisation is to reduce the problems of insert, update and delete
- Normalisation is done from an input perspective
- Normalisation is done from a forms perspective
- VIEW THE ENTIRE APPLICATION ON A PER-TRANSACTION BASIS, AND YOU NORMALISE EACH TRANSACTION SEPERATELY



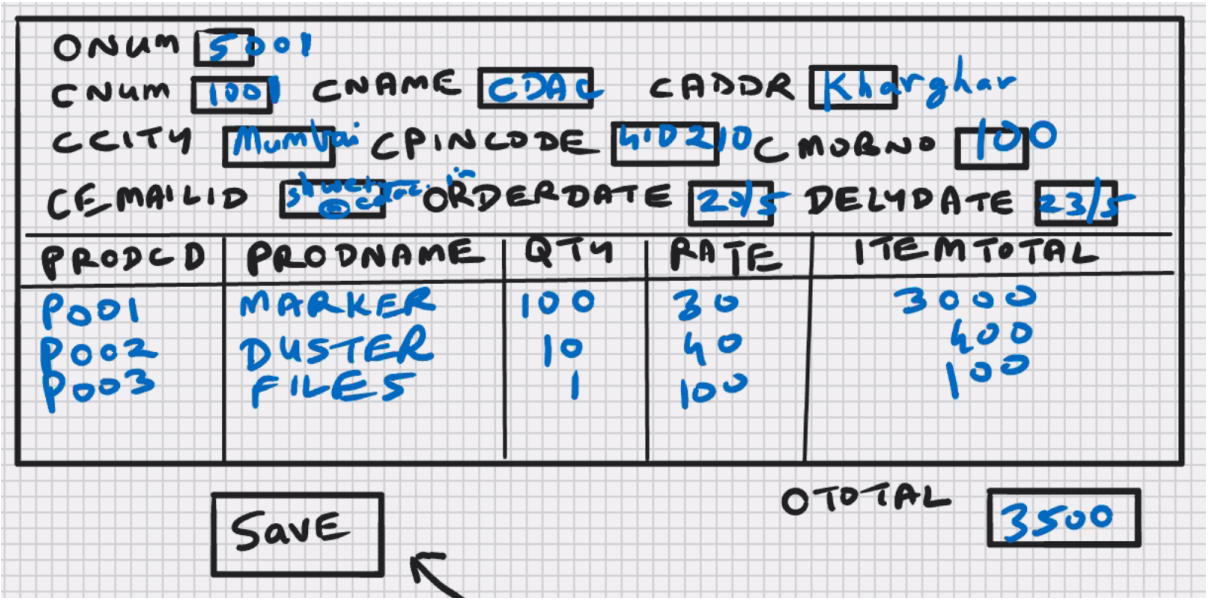
- E.g. CUSTOMER_PLACES_AN_ORDER, CUSTOMER_MAKES_PAYMENT, CUSTOMER_CANCELS_AN_ORDER, GOODS_ARE_DELIVERED, etc.

GETTING READY FOR NORMALISATION
(CUSTOMER_PLACES_AN_ORDER)

Onum
Cnum
Cname
Caddr
Ccity
Cpincode
Cmobno
Cemailid
Orderdate
Delydate
Prodcd
Prodname
Qty
Rate
Itemtotal
Ototal



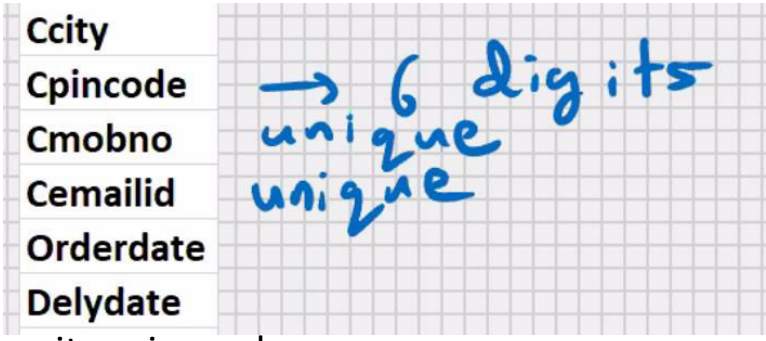
1. For a particular transaction, make a list of fields
2. Ask the client for sample data



3. With permission, involvement and approval of client :- Strive for atomicity (field can be divided into sub-fields, and sub-field can be divided into sub-sub-fields)
4. For every field, make a list of field properties
5. get user sign off

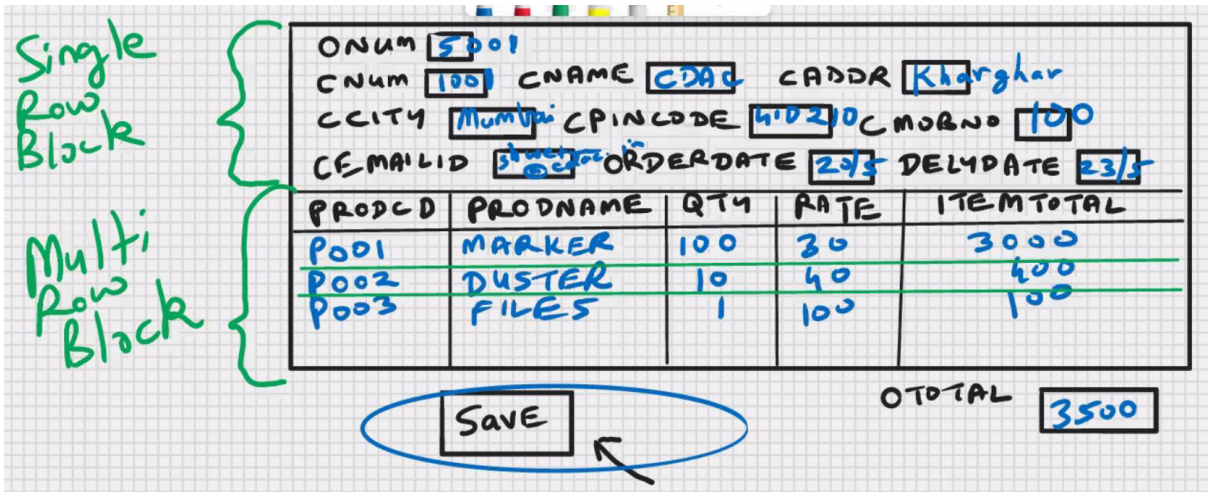
Caddr	→ alpha-numeric, max 200 characters, etc.
Ccity	
Cpincode	→ numeric, no decimal, 6 digits, fixed length, >0, mandatory, etc.

6. End of user involvement
7. Assign the datatype for each column
8. Assign not null, unique and check constraints
9. for all practical purposes, you can have a single table with all these columns



10. Key element will be Primary key of this table
11. If you don't have any key Element, then implement composite primary key
12. If you cannot implement composite primary key, then implement surrogate key

- At this point of time , data is in un-normalised form (UNF)
- UNF -> starting point of normalisation



- Remove the computed columns (e.g. itemtotal = qty*rate, Ototal = sum(qty*rate), etc)

9 STEPS OF NORMALISATION

1. remove the repeating group into a new table
2. Key element will be primary key of new Table
3. (this step may or may not be required) Add the primary key of the original table To the new table, to give you a Composite primary key

Onum			
Cnum			Prodcd
Cname			Prodname
Caddr			Qty
Ccity			Rate
Cpincode			
Cmobno			
Cemailid			
Orderdate			
Delydate			

Onum			Onum
Cnum			Prodcd
Cname			Prodname
Caddr			Qty
Ccity			Rate
Cpincode			
Cmobno			
Cemailid			
Orderdate			
Delydate			

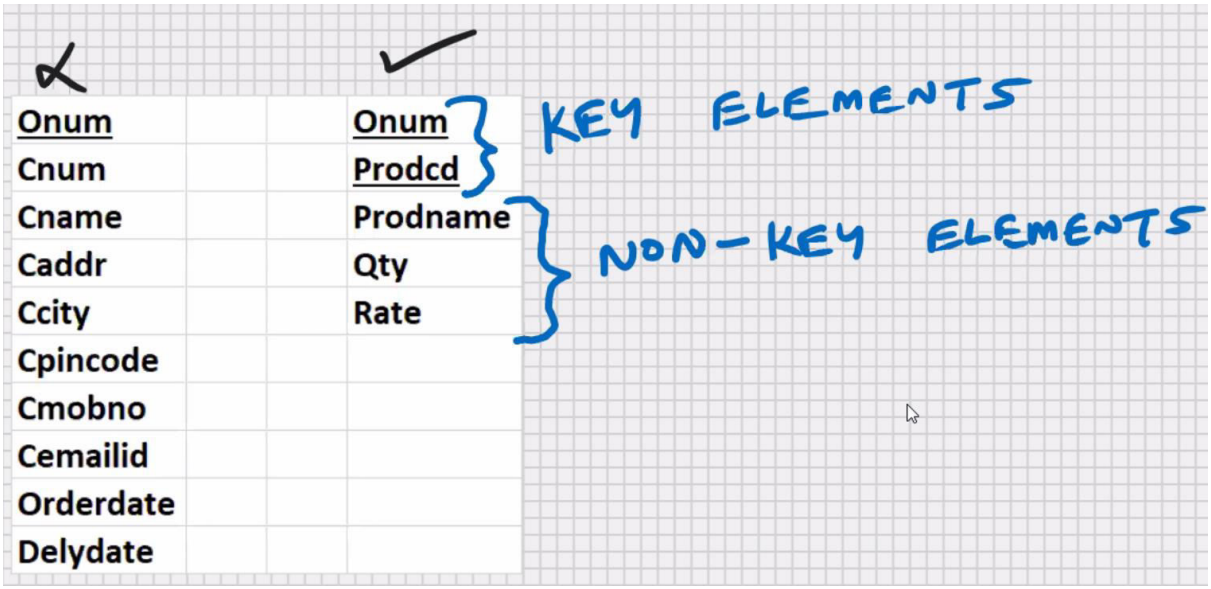
- Above 3 steps are to be repeated again and again infinitely till you cannot normalise any further

(FIRST NORMAL FORM) FNF /1NF(SINGLE NORMAL FORM) REPEATING GROUPS ARE REMOVED FROM TABLE DESIGN

- 1 : many relationship is always encountered here
- DEPT and EMP tables are in 1st Normal Form
- 25%

---

4. Only the tables with composite primary key are examined
5. Those non key elements that are not dependent on the entire composite key, they are to be removed into a new table
6. Key element on which originally dependent, it is to be added to the new table, and it will be the primary key of new table



Onum			Onum			
Cnum			Prodcd			Prodname
Cname			Qty			Rate
Caddr						
Ccity						
Cpincode						
Cmobno						
Cemailid						
Orderdate						
Delydate						

- **Above three steps are to be repeated again and again infinitely till you cannot normalise any further**

Second normal form (SNF) / double normal form (2NF)->

Every column is functionally dependent on primary key

- Known as functional dependency

- Functional dependency -> without primary key that column cannot work 67%

25% + 67% = 92%

Remaining 8 % i.e. less than 10 % we required third normal form

7. Only non key elements are examined for inter-dependencies
8. Interdependent columns are to be removed into new table

Onum			Onum			Prodcd
Cnum ✓			Prodcd			Prodname
Cname ✓			Qty			Rate
Caddr ✓						
Ccity ✓						
Cpincode ✓						
Cmobno ✓						
Cemailid ✓						
Orderdate						
Delydate						

Onum			Cnum			Onum			Prodcd
Orderdate			Cname			Prodcd			Prodname
Delydate			Caddr			Qty			Rate
Cnum			Ccity						
			Cpincode						
			Cmobno						
			Cemailid						

9. Key element will be the primary key of new table, and the primary key of new table, **it is to be retained in the original table for relationship purposes**
- **Above three steps are to be repeated again and again infinitely till you cannot normalise any further**

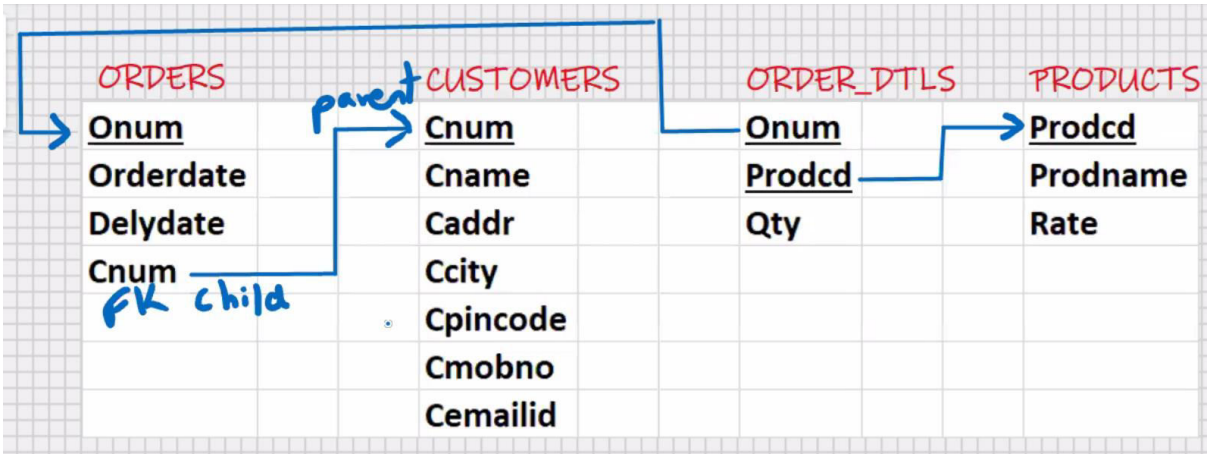
THIRD NORMAL FORM(TNF) / TRIPLE NORMAL FORM (3NF)

Transitive dependencies (inter-dependencies) (inter-dependent columns) are removed from table design

1NF- 1/4th
2NF- 2/3rd
3NF- 8%

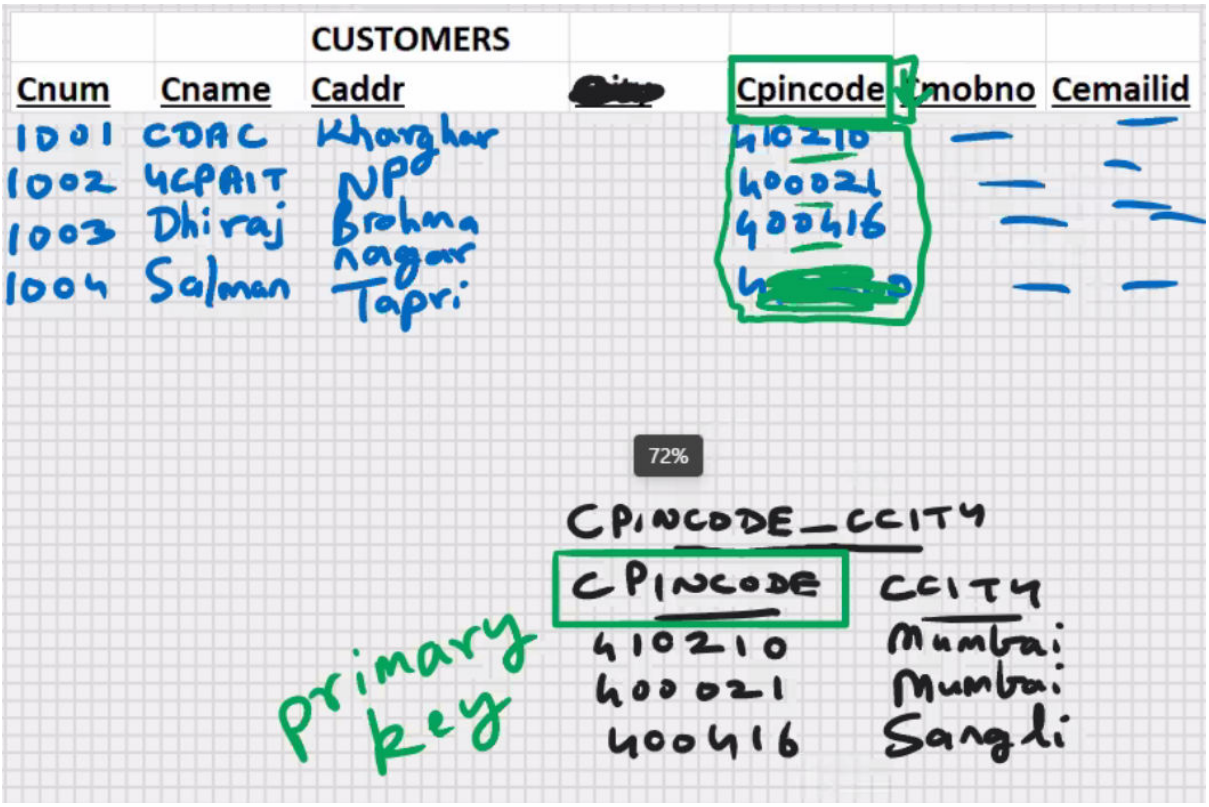
ORDERS	CUSTOMERS	ORDER_DTLS	PRODUCTS
Onum	**Cnum**	**Onum**	**Prodcd**
Orderdate	**Cname**	**Prodcd**	**Prodname**
Delydate	**Caddr**	**Qty**	**Rate**
Cnum	**Ccity**		
	Cpincode		
	Cmobno		
	Cemailid		

Normalisation
- What tables to create, structures, columns, datatypes, widths, and constraints
- Primary key is a by-product of normalisation

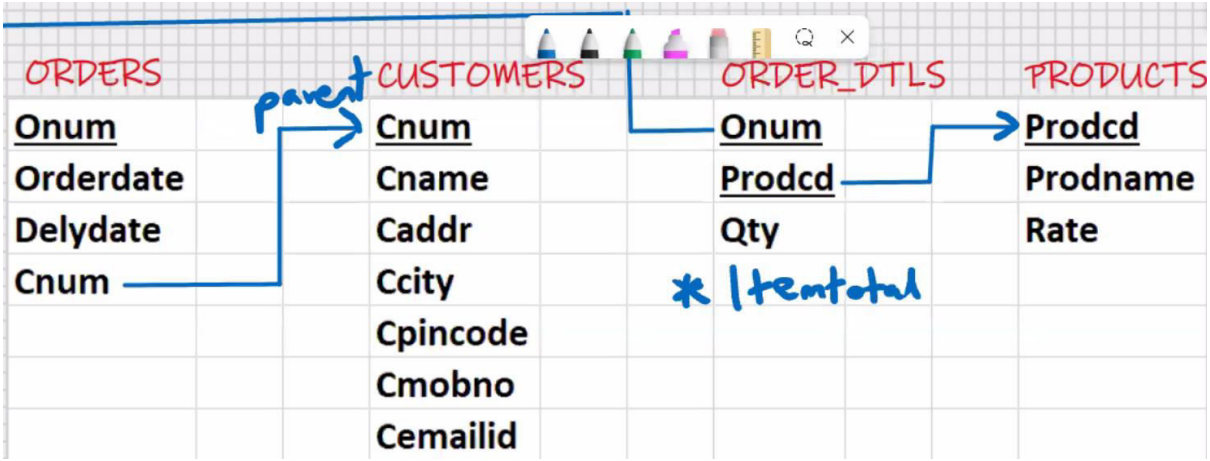


4th Normal form

- Extension of 3rd normal form
- Also known as boyce-codd normal form (BCNF)
- You may or may not implement 4th normal form
- Used to preserve the integrity of data
- Normally used on public network, e.g. internet



De-normalisation



- **if the data is large, if the SELECT statement is slow add an extra column to the table, to improve the performance, to make your SELECT statement work faster**
- **normally done for computed columns, expressions, summary columns, formula columns, function-based columns, etc.**

e.g. Itemtotal = Qty*Rate
Ototal = sum(itemtotal)

- **in some situations, you may want to create an extra table altogether and store the totals there**

Introduction to NoSQL (Not Only SQL)

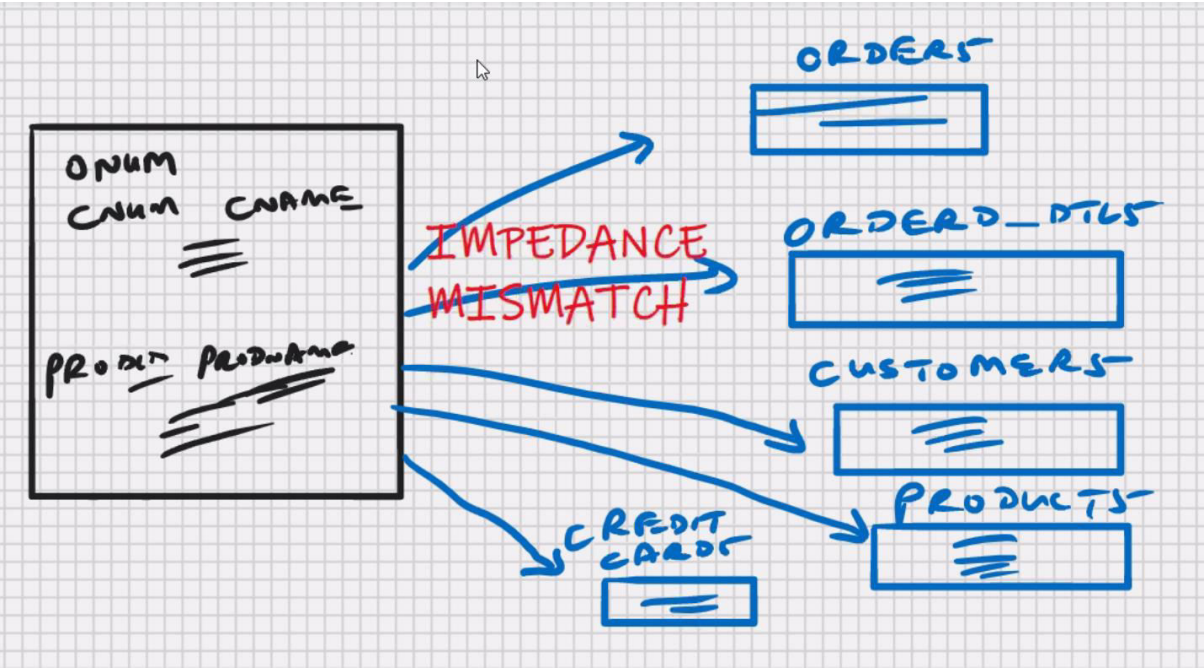
- New technology
- Not Only SQL (SQL is not the only language for database problems)
- Type of database management system

History:-

- Mid 1980's -> Rise of RDBMS

Benefits of RDBMS:-

a. Data persistence (Read and Write consistency) (achieved with the help of temporary new and old tables, and with the help of row locking)
b. SQL (common for all RDBMS)
c. Complex Transactions
d. Reporting tools (e.g. oracle reports, oracle apex, actuate, crystal reports, etc.)
e. Integration mechanism across applications



Problem with RDBMS model:-

One logical unit of fields in the the memory, at the time of saving, is splattered across multiple tables in the database

Solution:-

- In the mid 1990's -> Rise of Object databases (e.g. Oracle)

Oracle
RDBMS -> Relational DBMS
OODBMS -> Object Oriented DBMS
==========
OODBMS -> Object Oriented DBMS

Features of Object datatypes:-

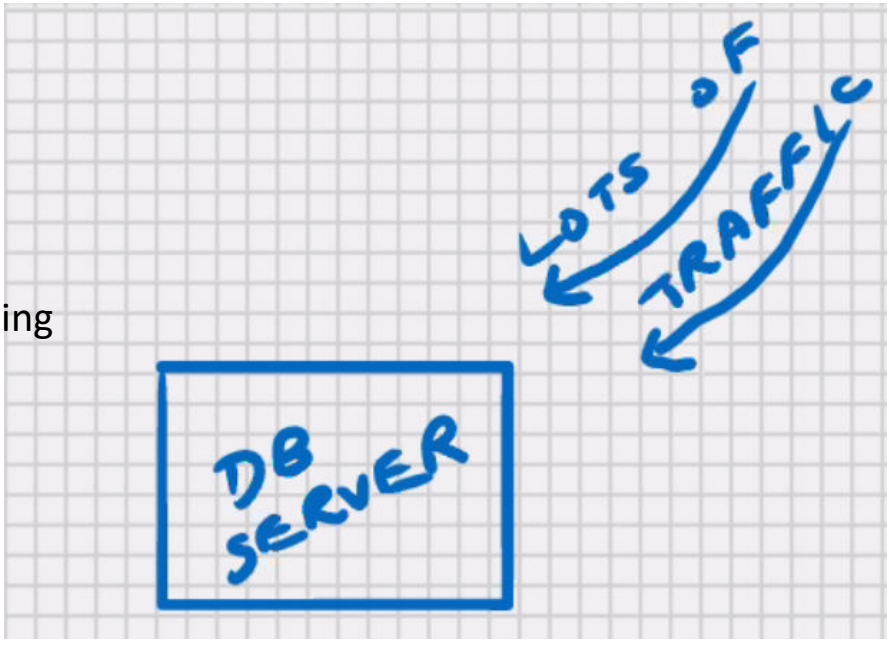
a. Abstract datatypes
b. Nested abstract datatypes
c. Object views
d. Bitmap Indexes
e. Index organized tables
f. Table partitioning across tablespaces / drives
g. Index partitioning across tablespaces / drives
h. Global and local indexes
i. Collectors
j. Nested tables
k. Varrays (varying arrays)
l. Varrays based on abstract datatypes
m. Instead of triggers
n. Advanced datatypes
o. Etc.

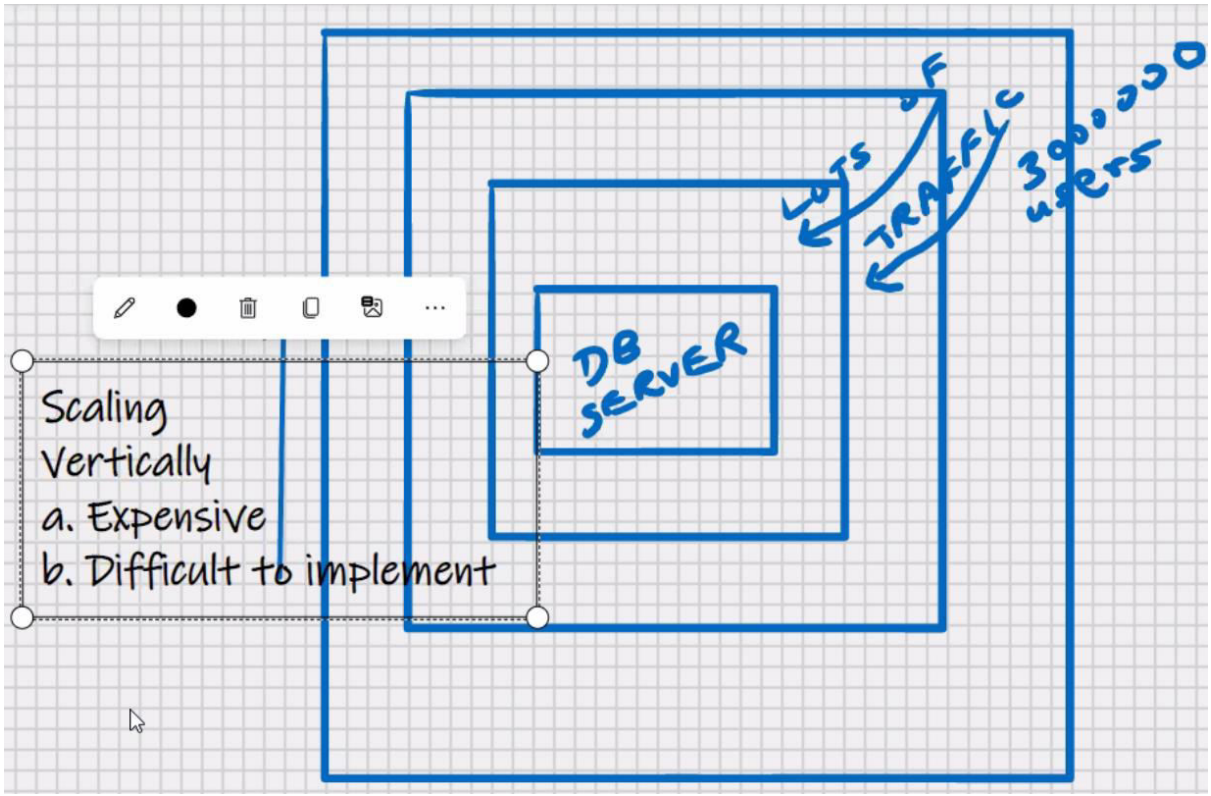
Early 2000s

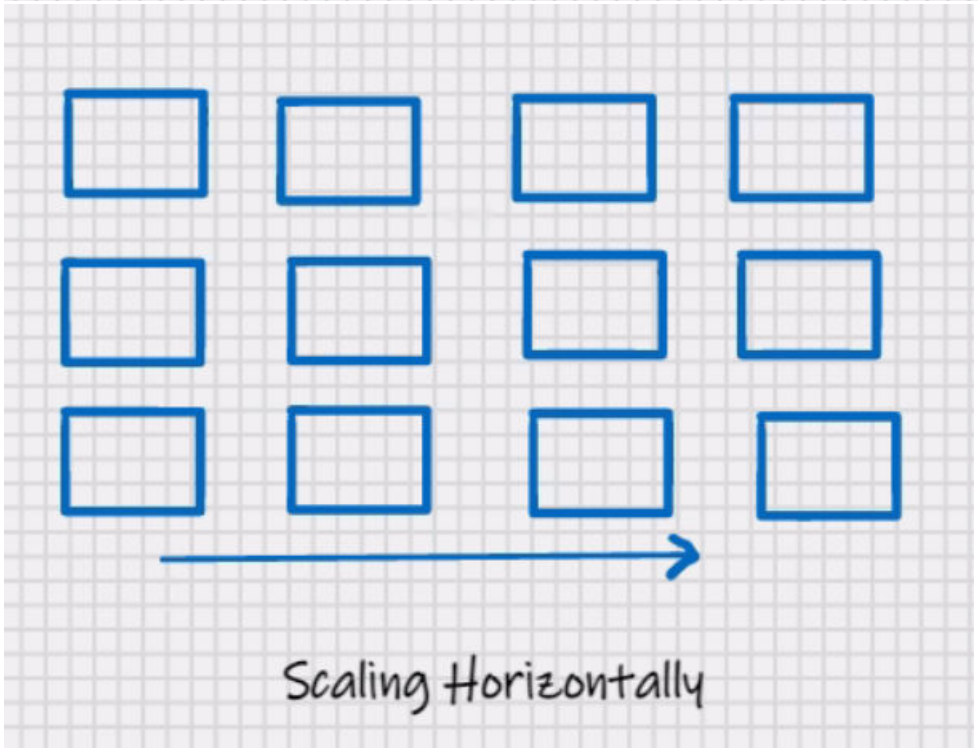
- Rise of the internet

Oracle 8i/9i- internet computing

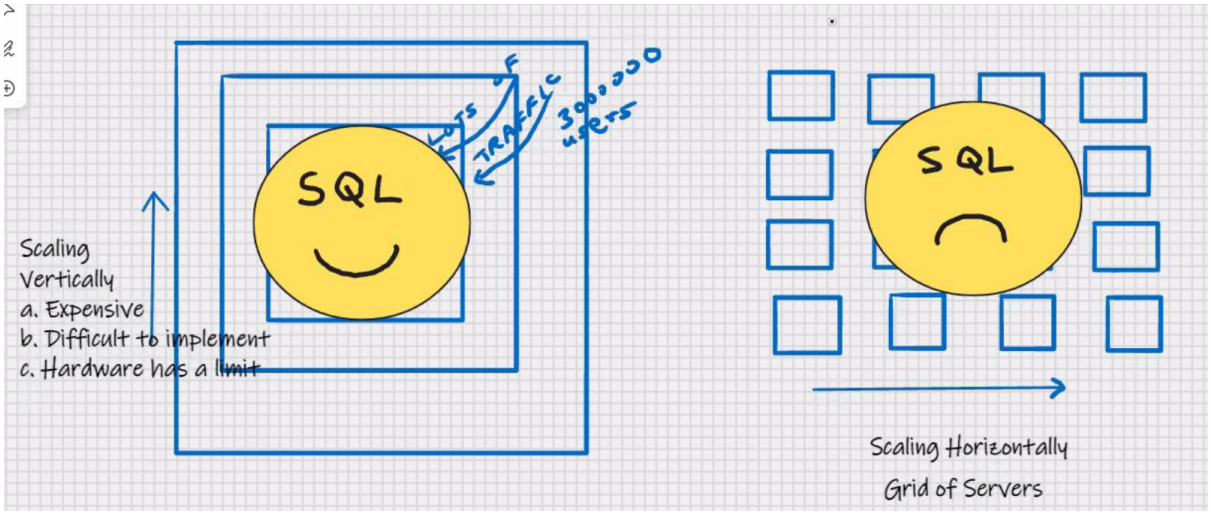
- High traffic website



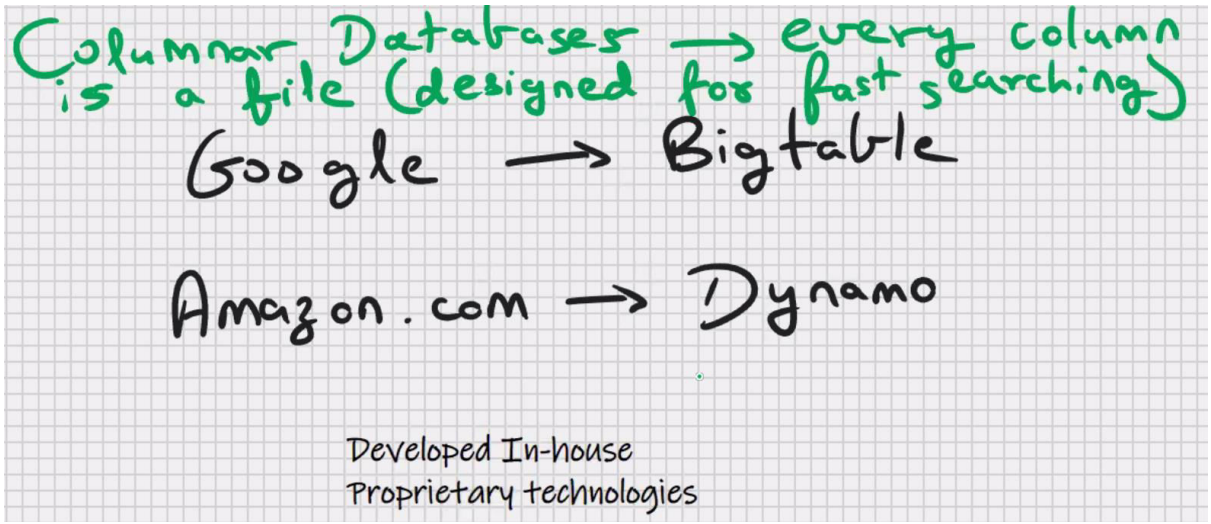




Solution :- scaling horizontally e.g. Oracle 10g/11g (grid computing)



To manage scaling horizontally: -
Oracle 12c/18c/19c/21c (cloud computing)
SAAS (Software as a service)
PAAS (platform as a service)
HAAS (hardware as a service)
IAAS (infrastructure as a service)



Birth of NoSQL

Twitter hashtag
#nosql

- To publicize his meeting

NoSQL meetup (soldout) :-
Thursday, 11th June, 2009 from 10 am to 5 pm
San francisco
Attendees :-
MongoDB
CouchDB relax
Project Voldemort A different database
Apache HBASE
Cassandra
Hypertable
Dynomite

Definition of NoSQL

- No definition of NoSQL

Characteristics of NoSQL:-

a. Open source
b. Cluster-friendly (ability to run on large clusters) (scaling horizontally)
c. 21st century web (high traffic websites)
d. Schema-less
e. Non-relational (no foreign key, no joins, etc.)

Various data models:

a. Document database
- Store documents
- E.g. MongoDB, Raven DB, couch DB relax

b. Column-family database
    - Every column is a file
    - E.g Cassandra, apached hbase
c. Graph database
    - Store graphs
    - E.g. neo4j NoSQL for the Enterprise
d. Key-value database

- Based on hashing algorithm
- From the value of the field it will generate the DB server HD storage address
- No searching involved and therefore the retrieval will be very fast
- Free HD space has to be allocated in advance
- Used for historical data (Typically a data warehousing application)
- Project Voldemort A distributed database, riak , redis

```
{"id":1001,
        "custid":7231, "custname":"CDAC":"address":"kharghar"}

{"id":1002,
        "custid":2317, "custname":"YCAPAIT":"address":"nariman point",
        "product_id":"P001","product_id":"P002"}

*       JSON-style documents (JavaScript Object Notation)
```

What is NoSQL?

NoSQL is type of DBMS:-

1. RDBMS
2. ORDBMS
3. OLAP (online analytical processing) (e.g. star-free, Cognos, Micro-strategy, icCube, etc.) (typical used in data warehousing applications)
4. NoSQL

Objectives of NoSQL?

NoSQL is focused to provide:-

1. Scalability
2. Performance
3. High availability

RDBMS vs NoSQL :-

RDBMS

1. More functionality
2. Less performance
3. Structured data
4. Tables

NoSQL

1. Less functionality
2. More performance
3. Structured and Unstructured data
4. Collections

NoSQL – What is Missing

1. No joins support
2. No complex transactions support
3. No constraint support

NoSQL- What is available

1. Query language (other than sql) (“Not only SQL”)
2. Fast performance
3. Horizontal scalability

When to use NoSQL

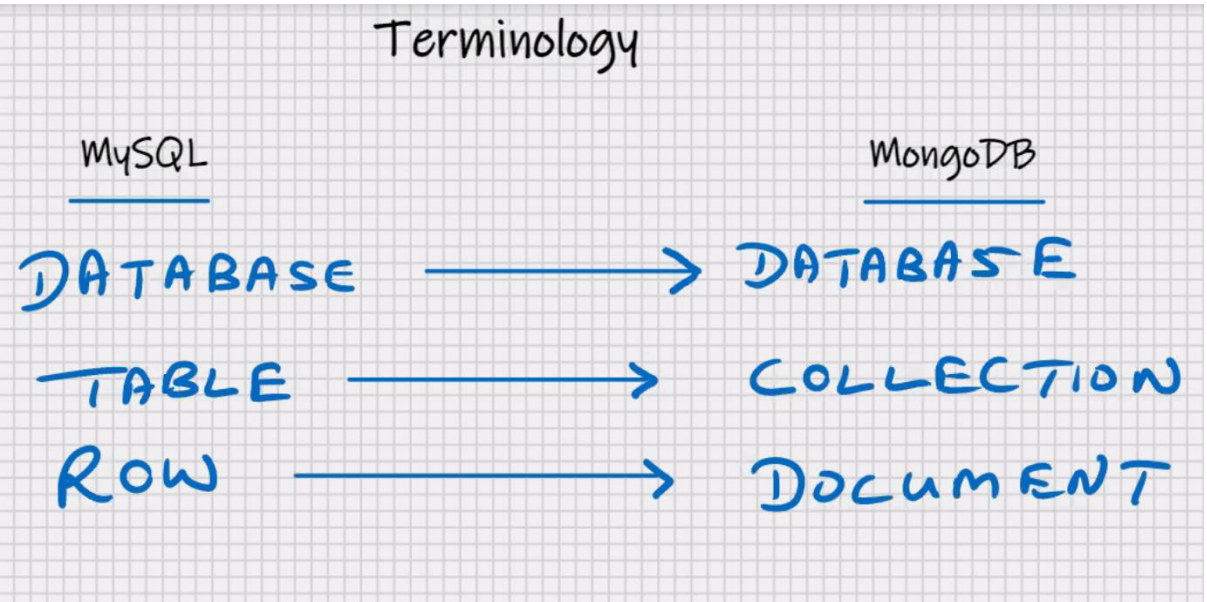
1. The ability to store and retrieve great quantities of data is important
2. The data is not structured or the structure is changing with time
3. Storing relationships between the elements is not important
4. Prototype of fast applications need to be developed
5. Dealing with growing lists of elements, e.g. twitter posts, internet server logs, blogs, etc
6. Constraints and validations is not required to be implemented in database

When not to use NoSQL

1. Complex transactions need to be handled by the database
2. Joins need to be handled by the database
3. Validations must be handled by the database

MongoDB

- Sponsored by 10gen



- 4 mb limit on document size
- You can have document within document
- No limit on nesting depths

JSON-style documents represented as BSON
JavaScript Object Notation style documents represented as Binary JavaScript Object notation
{“hello”:”world”}

```
\x16\x00\x00\x00\x02hello\x00\x06\x00\x00\x00world\x00\x00
```

Json.org/Bsonspec.org

Flexible "Schemas"

```
{"author":"himanshu","text":"..."}
{"author":"dishi","text":"...","tags":["mongodb"]}
```

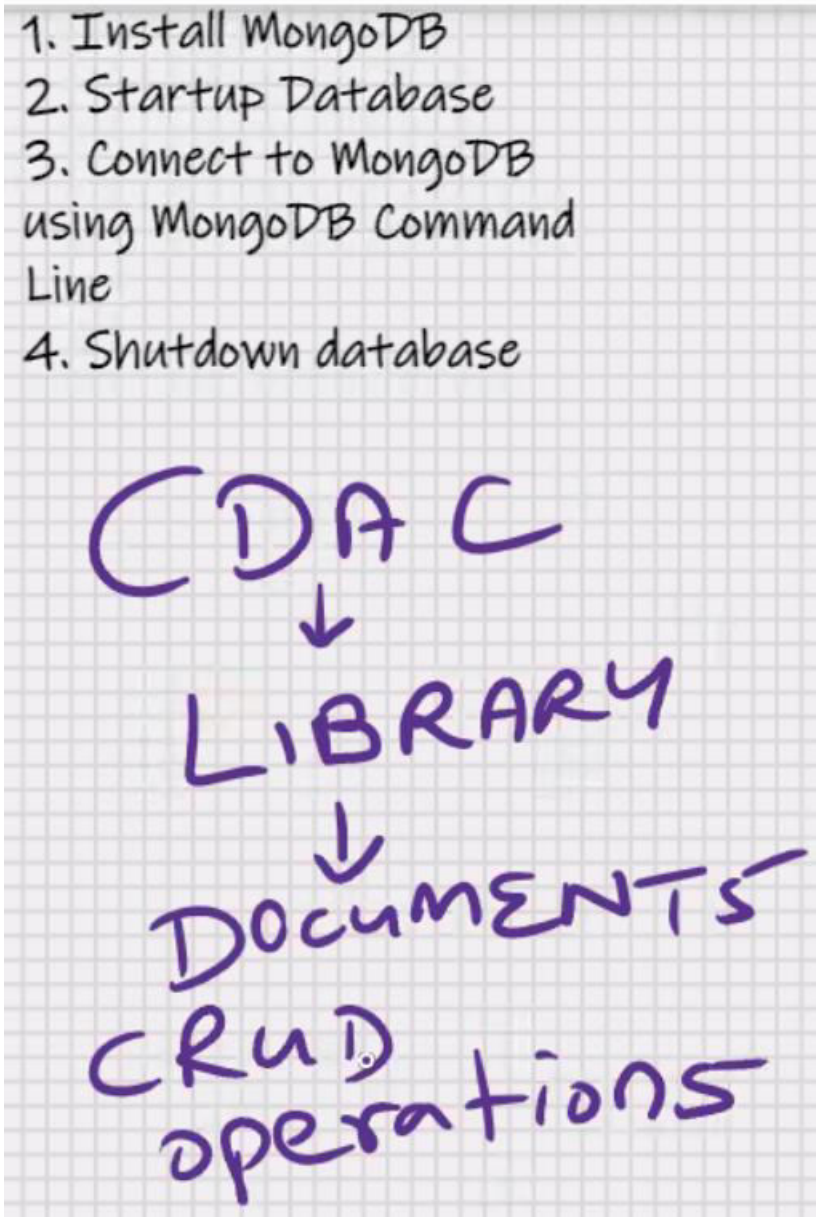
```
_id
*       every document has an _id field
*       equivalent to Primary key of RDBMS
*       drivers will add by default

Object("4bface1a2231316e04f3c434")
        timestamp
                machine id
                        process id
                                counter
```

- _id is lightweight occupying 12 bytes of storage
- Generated on client side to reduce the load on database server

Many Supported Languages/platforms

- Drivers for php, perl, c++, c#, .net, python, ruby, etc
- All major os 32/64 bit
- mongoDB v2.7 was the last version to support windows 32 bit
- all new versions only support 64 bit

Download MongoDB
http://www.mongodb.org



Install mongoDB

For win 10 64 bit

Run mongoDB setup

Click on Next
End-User License Agreement
Click on Next

Setup Type -> Complete
Click on Install
User Account Control -> Yes
Click on Finish

MongoDB gets installed in:-
C:\Program Files\MongoDB folder

Go to:-
C:\Program Files\MongoDB\Server\3.2\bin folder

Click on View (menu at the top) -> Filename extentions (checkbox) -> Check it

mongod.exe is required to start the MongoDB database
mongo.exe is required to start the MongoDB Command line

MongoDB requires a data directory to store all the data

Create folder C:\MongoDB\Data

To start the database:-

To add C:\Program Files\MongoDB\Server\3.2\bin to the Path variable in WIndows

Go to Windows Settings and search for environment varibles

Click on edit the system tab click on environment variables
In system variables you will see the path variables
Click on edit
Click on new
Add the following entry here:-

C:\Program Files\MongoDB\Server\3.2\bin
Click on Ok
Click on Ok
Restart your computer

To start the database:-

Go to Command Prompt:-
Go to C:\Program Files\MongoDB\Server\3.2\bin folder and type:-

mongod --dbpath C:\MongoDB\Data

If your path includes blank spaces, then enclose the entire path in double quotes

Do not close this Command prompt window

To connect to MongoDB
Open another Command prompt window

Go to C:\Program Files\MongoDB\Server\3.2\bin folder and type:-

mongo

to show databases:-

```
> show dbs
```

to create a new database:-

```
> use cdac
```

if cdac database already exists, then it will connent to it
if cdac database does not exist, then it will create it and it will connent to it

```
> db
```

```
> show dbs
```

we create cdac database, we connected to it, but it doesn't show in the list, because you should have at least one collection in it

to see list of collections:-

```
> show collections
```

to drop the database:-

```
> db.dropDatabase()
```

this will drop the selected database.
if you have not selected any database, then it will drop the 'test' database

to create collection:-

```
> db.createCollection("library")
```

to insert a document in library collection:-

```
> db.COLLECTION_NAME.insert(document)
```

```
> db.library.insert({title:'MongoDB programming',author:'Sameer',likes:100})
```

```
> db.library.find()
{ "_id" : ObjectId("6288f9f91d34fd95890b9177"), "title" : "MongoDB programming", "author" : "Sameer", "likes" : 100 }
>
```

to make it more presentable:-

```
> db.library.find().pretty()
```

we also have a findOne() method:-

```
> db.library.findOne()
```

Update document:-

```
> db.COLLECTION_NAME.update(SELECTION_CRITERIA,UPDATED_DATA)
```

Update document:-

```
> db.COLLECTION_NAME.update(SELECTION_CRITERIA,UPDATED_DATA)
> db.library.update({author:'Sameer'},{$set:{author:'Sameer Dehadrai'}})
```

Delete document:-

```
> db.COLLECTION_NAME.remove(DELETION_CRITERIA)
```

Let us delete the document that has 100 likes:-

```
> db.library.remove({likes:100})
```

To stop the MongoDB database:-

first exit the MongoDB shell (Command Line):-

Press Control+C in the terminal window where mongod instance is running