DBT (Database Technologies)

DAY1

Database Concepts
MySQL v5.7 (RDBMS)
Intro to Oracle v11g (ORDBMS) (Object Relational DBMS) (RDBMS + OODBMS)
Intro to MongoDB v3.2 (NoSQL DBMS) (Not Only SQL) (type of DBMS)

MySQL

Origin of the word Computer -> Computaire (French word) -> to compute/calculate

(input) (processing) (output)
Data -> Computer -> Information
(raw facts) (meaningful data)
22021984 (processed data)
(Data on whose basis you can take some action; or the management can make some decision)

Processing -> work done by the computer to convert the data into information

Database -> collection of LARGE amounts of data

DBMS -> Database Management System
DBMS -> readymade s/w that helps you to manage your data

ANSI definition of DBMS -> collection of programs that allows you to insert, update, delete, and process

Various DBMS available:

e.g.
MS Excel, dBase, FoxBASE, FoxPro, Clipper, DataEase, Dataflex, Advanced Revelation, DB Vista, Quattro Pro, etc.

DBMS (DB Mgmt System) vs RDBMS (Relational DataBase Management System)

DBMS (e.g. MS Excel, FoxPro, etc.)

a. Field
b. Record
c. File

1. Naming conventions (Nomenclature)
2. Relationship between 2 files is maintained programmatically
3. More programming
4. More time required for s/w development
5. High network traffic
6. Slow and expensive
7. Processing on Client machine
8. Client-Server architecture is not supported
9. File level locking

10. Not suitable for multi-user
11. Distributed Databases are not supported
12. No security (of data)
    DBMS is dependent on OS for security
    DBMS allows access to the data through the OS
    Security is not an in-built feature of DBMS

RDBMS (e.g. Oracle, MySQL etc.)

a. Column, Attribute, Key
b. Row, Tuple, Entity
c. Table, Relation, Entity class

1. Naming conventions (Nomenclature)
2. Relationship between 2 tables can be specified at the time of table creation (e.g. Foreign key constraint) 3. Less programming
4. Less time required for s/w development
5. Low network traffic
6. Faster (in terms of network speed) and cheaper (in terms of hardware cost, network cost, infrastructure cost)
7. Processing on Server machine (known as Client-Server architecture)
8. Most of the RDBMS support Client-Server architecture
9. Row level locking (internally table is not a file, internally every row is a file)
10. Suitable for multi-user
11. Most of the RDBMS support Distributed Databases (Banking system is an example of Distributed Databases)
12. Multiple levels of security

    a. Logging in security
       (MySQL database username and password)
    b. Command level security
       (permission to issue MySQL commands)
       (e.g. create table, create function, create user, etc.)
    c. Object level security
       (to access the tables and other objects of other users)

Various RDBMS available:

Informix (fastest in terms of processing speed)
Oracle
Sybase
MS SQL Server
Ingres
Postgres
Unify Non-Stop
DB2
CICS
TELON
IDMS MS Access
Paradox Vatcom SQL
MySQL etc.

Oracle

- most popular (because it has best the best tools for s/w development)
- (makes programming very easy)
- product of Oracle Corporation (founded in 1977)
- #1 largest overall s/w company in the world
- #1 largest DB s/w company in the world
- 63% of world commercial DB market in Client-Server environment
- 86% of world commercial DB market in the Internet environment
- works on 113 OS
- 10/10 0f top 10 companies in the world use Oracle

Sybase

- going down
- recently acquired by SAP

MS SQL Server

- good RDBMS from Microsoft (17% of world commercial DB market)
- only works with Windows OS

Open-source free RDBMS:

(character based) (text based) :

* Ingres
* Postgres
* Unify
* Non-Stop

DB server has to be a mainframe (super computer) :- DB2 (good RDBMS from IBM)

* CICS
* TELON
* IDMS

Single-user PC based RDBMS: - MS Access
Paradox
Vatcom SQL

MySQL

* MySQL was launched by a Swedish company in 1995
* Its name is a combination of "My", the name of co-founder Michael Widenius' daughter, and "SQL"
* MySQL is an open-source RDBMS
* MySQL was initially free
* Most widely used open-source RDBMS
* Part of the widely used LAMP open-source web application software stack (and other "AMP" stacks)
* Free-software open-source projects that require a RDBMS often use MySQL
* Occupies 42% of free database s/w market
* WordPress, Facebook, Twitter, Flickr, YouTube, Google (though not for searches), WhatsApp, Instagram, etc.
* Sun Microsystems acquired MySQL in 2008

* Oracle Corporation acquired Sun Microsystems in 2010

Various s/w development tools of MySQL:

SQL

* Structured Query Language
* Commonly pronounced as "Sequel"
* Create, Drop, Alter
  Insert, Update, Delete
  Grant, Revoke, Select
* Conforms to ANSI standards (e.g. 1 character = 1 Byte)
* Conforms to ISO standards (for QA)
* Common for all RDBMS
* Initially founded by IBM (1975-77)
* Initially known as RQBE (Relational Query by Example)
* IBM gave RQBE free of cost to ANSI
* ANSI renamed RQBE to SQL
* Now controlled by ANSI
* In 2005, source code of SQL was rewritten in Java (100%)

MySQL command line client

* MySQL client software
* Used for running SQL commands, MySQL commands, MySQL PL programs, etc.
* Character based (text based)
* Interface with database

MySQL Workbench

* MySQL client software
* Used for running SQL commands, MySQL commands, MySQL PL programs, etc.
* GUI based (Graphical User Interface) interface with database

MySQL PL

* MySQL Programming Language
* Programming language from MySQL
* Used for database programming
  e.g. HRA_CALC, TAX_CALC, ATTENDANCE_CALC, etc.

MySQL Connectors

* for database connectivity (JDBC, ODBC, Python, C, C++, etc.)

MySQL for Excel

* import, export, and edit MySQL data using MS Excel

MySQL Notifier

* Start-up and Shutdown the MySQL database

MySQL Enterprise Backup

* export and import of table data
* used to take backups and restore from the backups

MySQL Enterprise High Availability

* for replication (also known as data mirroring) concept of standby database

MySQL Enterprise Encryption

* used to encrypt the table data

MySQL Enterprise Manager

* for performance monitoring, and performance tuning

MySQL Query Analyzer

* for query tuning

MySQL SQL

Common for all RDBMS:

- 4 sub-divisions of SQL:

DDL (Data Definition Language) :- (Create, Drop, Alter)
DML (Data Manipulation Language) :-(Insert, Update, Delete)
DCL (Data Control Language) :- (Grant, Revoke)
DQL (Data Query Language) :- (Select)

Extra in Oracle RDBMS and MySQL

RDBMS: - Not an ANSI standard: - 5th

component of SQL: -

DTL/TCL (Data Transaction Language) / (Transaction Control Language)
(Commit, Rollback, Savepoint)
DDL (Rename, Truncate)

Extra in Oracle RDBMS only:

DML (Merge, Upsert)

Rules for table names, column names, and variable names:

* **Oracle** : Max 30 characters **MySQL** : Max 64 characters
* A - Z, a - z, 0-9 allowed
* Has to begin with an alphabet
* Special characters $, #, allowed
* In MySQL, to use reserved characters such as # in table name and

* Column name, enclose it in backquotes * ` ` backquotes e.g. `EMP#`
* 134 reserved words not allowed

Datatypes:-

Char :

- (allows any character) (max upto 255 characters) (default width 1)
- (wastage of HD space) (searching and retrieval is very fast)
- e.g. ROLL NO, EMPNO, PANNO, etc.

Varchar :

- (allows any character) (max upto 65,535 characters) (64 KB - 1)
- (no default width) (width has to be specified) (conserve on HD space)
- (searching and retrieval is compromised)
  e.g. ENAME, ADDRESS, CITY, etc.

Day 2

DATATYPES:-

Text

Tinytext (allows any character) (max upto 255 characters)

Text (allows any character) (max upto 65,535 characters)

Mediumtext (allows any character) (max upto 16,777,215 characters) (16 MB)

Longtext (allows any character) (max upto 4,294,967,295 characters) (4 GB)

* all of the above are stored outside the row
* stored outside the table
* stored away from the table
* MySQL maintains a LOCATOR (HD pointer) from the table row to the text data
* this datatype is used for those columns that have a large amount of text and will not be used for searching
* e.g. REMARKS, COMMENTS, EXPERIENCE, RESUME, FEEDBACK, REVIEW, etc.
* width does not have to be specified for all of the above datatypes

Binary (fixed length binary string) (max upto 255 Bytes of binary data) (e.g. small images)
(e.g. BARCODES, PICTURE_CODES, QR_CODES, FINGERPRINTS, SIGNATURES, etc.) (width need not be specified)

Varbinary (variable length binary string) (max upto 65,535 Bytes of binary data)
(e.g. STICKERS, EMOTICONS, EMOJIS, ICONS, etc.) (no default width) (width has to be specified)

* both of the above are stored as character strings of 1's and 0's

Blob -> Binary Large Object

Tinyblob (max upto 255 Bytes of binary data)
Blob (max upto 65,535 Bytes of binary data)
Mediumblob (max upto 16,777,215 Bytes of binary data)
Longblob (max upto 4,294,967,295 Bytes of binary data)

* all of the above are stored outside the row
* outside the table
* MySQL maintains a LOCATOR from the table row to the Blob data
* used for those columns that are meant for display purposes and not for searching purposes
* width does not have to be specified in all of the above datatypes
* e.g. PHOTOGRAPHS, WALLPAPERS, SOUND, MUSIC, VIDEOS
* Blob is the multimedia datatype of MySQL

Integer types (**Exact value**) :

Signed or Unsigned: - by default it is signed

Tinyint (occupies 1 Byte of storage)
Smallint (occupies 2 Bytes of storage)
Mediumint (occupies 3 Bytes of storage)
Int (occupies 4 Bytes of storage)
Bigint (occupies 8 Bytes of storage)

* e.g. age tinyint unsigned

Floating Point types:-

(Approximate value) :-

Float :- (single precision) (up to 7 decimals)
Double :- upto 15 decimals

Decimal (stores double as a string)
(e.g. "653.7") (max number of digits is 65)
(used when it is important to preserve exact precision, for example with monetary data)

Boolean

- (True and False evaluate to 1 and 0 respectively)
  e.g. MARITAL STATUS boolean

* can insert true, false, 1, or 0
* output will display 1 or 0

Date and Time Datatypes:

Date ('YYYY-MM-DD' is the default date format)
('1000-01-01' to '9999-12-31')
(specifying all 4 digits of year is optional)
e.g. '21-06-22'
(year values in the range 70-99 are converted to 1970-1999)
(year values in the range 00-69 are converted to 2000-2069)

Why 1970 is the cut-off year?
Unix was originally developed in the 60s and 70s so the "start" of Unix Time was set to January 1st 1970 at midnight GMT (Greenwich Mean Time) - this date/time was assigned the Unix Time value of 0

date1-date2 -> returns number of days between the 2 dates

'1000-01-01' -> 1
'1000-01-02' -> 2
'1000-01-03' -> 3

'2021-06-22' -> 2456173 (number of days since '1000-01-01')
internally date is stored a fixed-length number
Date occupies 7bytes of storage

Time
('hh:mm:ss') or ('HHH:MM:SS')

(time values may range from 1-838:59:59' to '838:59:59')

Datetime ('YYYY-MM-DD hh:mm:ss')
('1000-01-01 00:00:00' to '9999-12-31 23:59:59')

datetime1-datetime2 -> returns number of days, remainder hours, remainder minutes, remainder seconds between the two

Year (YYYY) (1901 to 2155)

* max 4096 columns per table provided the row size <= 65,535 Bytes
* no limit on number of rows per table provided the table size <= 64 Terabytes

COMMAND to CREATE TABLE:-

******(commands are case insensitive)**

```
create table emp (empno char (4), ename varchar (25),sal float, city varchar (15), dob date );
```

"**;**" is known as terminator (denotes the end of command)

COMMAND to INSERT into the TABLE:-

(one row at a time)
insert into emp values ('1', 'Aakash', 5000, 'Mumbai', '1995-10-01');
*******for char,varchar & date use ' '**

insert into emp (empno, sal, ename, city, dob)
values ('2', 6000, 'Mahesh', 'Mirzapur', '1991-06-08'); -> **recommended**

a. flexible
b. readable
c. in future if you alter the table, if you add a column, it will continue to work

insert into emp (empno, sal) values ('3', 7000);

insert into emp values ('4', 'Ajay'); -> error

insert into emp values ('4', 'Ajay', null, null, null);

insert into emp values ('5', null, 5000, null, null);

******** null means nothing and null has ASCII value 0**

* special treatment given to null value in all RDBMS**:-(independent of datatype)**
* null value occupies only 1 byte of storage
* if row is ending with null values,those columns will not occupy space
* its recommended that those columns that are likely to have a large number of null values should preferably be specified at the end of the table structure; to conserve on HD space

(insert multiple rows simultaneously)

insert into emp values ('1', 'A', 5000, 'Mumbai', '1990-04-05'),('2', 'B', 5000, 'Delhi', '1991-06-15');

insert into emp (empno, sal) values ('1', 5000),('2', 6000),('3', 7000);

SELECT COMMAND to Display:-

select * from table_name;
Here, **"*"** is known as **metacharacter(**all columns)

1)Read
2)Compile (convert into machine lang)
3)Plan (go to server HD search for table and return the output to my machine)
4)Excecute

To restrict Columns:-
select empno,ename from emp;

(searching takes place in DB server HD)

* position of columns in SELECT statement will determine the position of columns in the output (as per user requirements)

To restrict Rows:-
(using WHERE clause)

```
select * from emp where deptno = 10;
```

* 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

```
select * from emp where sal > 2000;
```

Relational Operators:-

1. >
2. >=
3. <
4. <=
5. != or <>
6. =

```
select * from emp where sal > 2000 and sal < 3000;
```

Logical Operators:-
1. NOT
2. AND
3. OR

```
select * from emp where deptno = 10 or sal > 2000 and sal < 3000;
```

```
select * from emp where (deptno = 10 or sal > 2000) and sal < 3000;
```

```
select * from emp where job = 'MANAGER';
```

* In Oracle & MySQL, at the time of inserting, data is case-sensitive
* In Oracle, queries are case-sensitive (more secure)
* In MySQL, queries are case-insensitive (more user-friendly)

```
select * from emp where job = 'MANAGER' or job = 'CLERK';
```

```
select * from emp where job = 'MANAGER' and job = 'CLERK';
```
(no rows selected)

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

sal*12 -> computed column, derived column, virtual column, fake column, pseudo column
* Processing/calculation takes place in server RAM

Arithmetic Operators:-

1. ( ) grouping
2. ** exponential e.g sal**3 means (sal^3)

**** doesn't work in MySQL**

"**" works in Oracle PL/SQL

In MySQL, if you want to use exponential then u have to use power function

3. / division
4. * multiplication
5. + addition
6. - substraction

alias (used to display new name of column)

```
select ename, sal, sal*12 as "ANNUAL" from emp;
```

```
select ename, sal, sal*12 "ANNUAL" from emp;
```

as -> ANSI SQL
as -> Optional in MySQL and Oracle

* you cannot use alias in an expression

distinct (keyword)

```
select distinct job from emp;
```

* whenever you use DISTINCT, sorting takes place in server RAM
* if you have a large number of rows, thensorting is one operation which is always slows down the processing

```
select distinct job, ename from emp;
```

performs operation on both job & ename

Installation:-

When you install MySQL, 2 users are autocatically created:

1. mysql.sys
* owner of database
* owner of system tables
* startup database, shutdown database, perform recovery,etc.

2. root
* has Database Administrator DBA privileges
* create users, assign privileges, configure database, perform planning, monitoring, tunning, take backups,etc.

DAY 3

DBMS -

Data is stored sequentially

RDBMS –

Data is stored randomly anywhere(each row is file) mixed with another data

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

* rows inside a table are not sequentially
* rows inside a table are scattered (fragmented) all over the DB server HD
* when you INSERT into a table wherever it finds the free space in the DB server HD, it will store the row there
* the reason that RDBMS does this is to speed up the INSERT statement
* when you SELECT from a table, the order of rows in the output depends on the row address (searching is always sequential)
* when you SELECT from a table, the order of rows in the output will always be in ascending order of row address
* when you UPDATE a row, if the row length is increasing, the row address MAY change (it's only in the case of VARCHAR that row length may increase)
* hence it's not possible to see the first 'N' rows inserted in a table or the last 'N' rows inserted in a table

ORDER BY clause:- (used for sorting)

select deptno, job, ename, sal, hiredate from emp
order by ename; (by name)

select deptno, job, ename, sal, hiredate from emp order by asc; (asecnding)

select deptno, job, ename, sal, hiredate from emp order by desc; **(descending)**

asc -> by default
desc

select deptno, job, ename, sal, hiredate from emp order by deptno; **(by deptno**)

select deptno, job, ename, sal, hiredate from emp order by deptno,job;
(first it will sort on basis of deptno if deptno is same then it will sort on basis of job)

select deptno, job, ename, sal, hiredate from emp order by deptno desc,job desc;

* no upper limit on number of columns in ORDER BY clause

select.....order by country, state, district, city;

* if you have large number of rows in the table, and large numbe of columns in ORDER BY clause, the SELECT statement will be slow

select ename, sal*12 from emp;

select ename, sal*12 from emp order by sal*12;
select ename, sal*12 annual from emp order by annual;

* **ORDER BY clause is the LAST clause in SELECT statement**

select ename, sal*12 "Annual Salary" from emp order by "Annual Salary";

select ename, sal*12 "Annual Salary" from emp order by 2; (2 is column no in select statement)

ORDER BY clause is the LAST clause in SELECT statement

select ename, sal*12 "Annual Salary" from emp order by "Annual Salary";

select * from emp order by 2;

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

Blank padded comparision semantics:-

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

select * from emp where ename >= 'A' and ename < 'B';

Special Operators:- (Like, Between)

Like:-

select * from emp where ename like 'A%';

Solution for case-insensitive query in Oracle:-

select * from emp where ename like 'A%' or ename like 'a%';

Wildcards (used for pattern matching)
% any character and any number of characters
_ any 1 character

select * from emp where ename = 'A%';

select * from emp where ename like '%A'; (returns values ending with A)

select * from emp where ename like '%A%'; (returns values containing A)

select * from emp where ename like '__A%'; (returns values containing A as 3rd letter)

select * from emp where ename like '____'; (returns values containing 4 letters)

select * from emp where sal >= 2000 and sal <= 3000;

Between:-

select * from emp where sal between 2000 and 3000; -> recommended
* easier to write
* works faster

select * from emp where sal not between 2000 and 3000;

select * from emp where sal < 2000 or sal > 3000;

select * from emp where hiredate between '2020-01-01' and '20202-12-31';

select * from emp where hiredate >= '2020-01-01' and hiredate <= '20202-12-31';

select * from emp where ename between 'A' and 'F';

select * from emp where ename >= 'A' and ename <= 'F';

---

select * from emp where deptno = 10 or deptno = 20 or deptno = 40;

select * from emp where deptno = any (10,20,40); -> FASTER

select * from emp where deptno in (10,20,40); -> FASTEST

* **IN operator is faster than ANY operator**
* **ANY operator is more powerful than IN operator**
* with IN, you can only check for IN and NOT IN whereas with ANY, you can check for =ANY, !=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

select from emp where city in ('Mumbai', 'Delhi');

* ANY operator woks dirctly in Oracle
* **ANY operator does not work directly in MySQL(But Exception is there "ANY" operatoer used in MySQL within Sub Queries)**
* in MySQL, ANY operator has to be used with sub-query
* in MySQL, use the IN operator

DDL -> create, drop
DML -> insert, update
DQL> select .... *, coll, co12, WHERE clause, Relational, Logical, Arithmetic, Special Operators, Computed column, Alias, ORDER BY clause

UPDATE

```
update emp set sal = 10000 where empno = 1;

update emp set sal = sal + sal*0.4 where empno = 1;

update emp set sal = 10000, city = 'Pune' where empno = 1;

update emp set sal = 10000 where city = 'Mumbai';

update emp set sal = 10000 , city = 'Pune' where city = 'Mumbai';
```

* you can UPDATE multiple rows and multiple columns simultaneously, but you can UPDATE only 1 table at a time
* if you want to UPDATE multiple tables simultaneously, it is not possible; you will require a separate UPDATE command for each table

update emp set sal = 10000; (performs operation on whole table)

DELETE

```
delete from emp where empno = 1;
```

FROM -> ANSI SQL
FROM -> optional in Oracle, but it is required in MySQL

```
delete from emp where city = 'Mumbai';

delete from emp;
```

(all rows will be deleted, empty table)

DROP

drop table emp;(whole table ROWs will be deleted But Table Exists)

* you cannot use WHERE clause with DROP table
* if you want to drop multiple tables, then you will have to drop each table separately
* a separate DROP table command would be required for each table

* **UPDATE and DELETE commands without WHERE clause will not be allowed in MySQL Workbench**

to issue UPDATE and DELETE commands without WHERE clause in MySQL Workbench: -
Click on Edit (menu at the top) -> Preference -> SQL Editor -> "Safe Updates" checkbox at the bottom -> uncheck it -> click on Ok
Click on Query (menu at the top) -> Reconnect to server

DAY 4

TRANSACTION PROCESSING

COMMIT: -

* Commit will save all the DML 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

Total Work done = T1 + T2 + T3 + ... + Tn;

* when to issue the Commit depends upon the logical scope of Work

commit work;

* work is ANSI SQL
* **work is optional** in Oracle and MySQL

ROLLBACK: -

rollback work;

* Rollback will undo all the DML changes since the last committed state

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

* **only the DML** commands are affected by Rollback and Commit
* any DDL command automatically commits
* when you exit from SQL*Plus, it automatically commits
* any kind of power failure, network failure, system failure, window close, improper exit from SQL, etc.; your last uncommitted Transaction is automatically Rolled back

SAVEPOINT: -

savepoint somename; (somename is max upto 30 chars)

* you can Rollback to a Savepoint
* Savepoint is a point within a transaction (similar to bookmark)
* YOU CANNOT COMMIT TO A SAVEPOINT
* Commit will save all the DML changes since the last committed state
* when you Rollback or Commit, the intermediate Savepoints are automatically cleared
* if you to use those Savepoints again, you will have to reissue them in your Work

ROLLBACK to SAVEPOINT: -

rollback work to pqr;

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

rollback to pqr;

* Savepoint is a sub-unit of Work
* within a Transaction, you can have 2 Savepoints with the same name; the latest Savepoint overwrites the previous one; the older Savepoint no longer exists

To try out Rollback, Commot, Savepoint in MySQL Workbench:

Click on Query (menu at the top) -> Auto-Commit Transactions - Uncheck it

READ and WRITE Consistency: -

* In a multi-user environment, when you SELECT from a table, you can view: -
* only the committed data of all users

plus
changes made by you

ROW LOCKING: -

* when you UPDATE or DELETE a row, that row is automatically locked for other users
* ROW LOCKING IS AUTOMATIC IN MYSQL AND ORACLE
* when you UPDATE or DELETE a row, that row becomes READ ONLY for other users
* other users can SELECT from that table; they will view the old data before your changes
* other users can INSERT rows into that table
* other users can UPDATE or DELETE "other" rows of 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**

OPTIMISTIC ROW LOCKING: -

* automatic row locking mechanism in MySQL and Oracle

To try out row locking in MySQL Workbench: -

Click on Query (menu at the top) -> New tab to current server -> click on it

* now you will have 2 query windows to try out row locking

To abort the operation (to exit from the Request queue) -> Click on query (menu at the top) -> Click on Stop

PESSIMISTIC ROW LOCKING: -

* you manually lock the rows BEFORE issuing UPDATE or DELETE
* to lock the rows manually you require SELECT statement with a FOR UPDATE clause

select * from emp where deptno = 10 for update;

* when you try to lock the row manually, if some other user has locked the same row before you, then by default your request will wait in the Request Queue

select * from emp where deptno = 10 for update wait; -> (by default)

select * from emp where deptno = 10 for update wait 60; -> (time in SECONDS)

select * from emp where deptno = 10 for update nowait;

* **WAIT/NO WAIT options are not available in MySQL**
* LOCKS ARE AUTOMATICALLY RELEASED WHEN YOU ROLLBACK OR COMMIT

FUNCTIONS: -

EMP

FNAME	LNAME
Arun	Purun
Tarun	Arun
Sirun	Kirun
Nutan	Purun

|| CONCATENATION Operator: -

select fname||lname from emp;

OUTPUT:- fname||lname

```
------------
ArunPurun
TarunArun
SirunKirun
NutanPurun
```

select fname||' '||lname from emp;

OUTPUT:- fname||' '||lname

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

select fname||', '||lname from emp;

OUTPUT:- fname||', '||lname

```
-----------------
Arun, Purun
Tarun, Arun
Sirun, Kirun
Nutan, Purun
```

select 'Mr. '||fname||' '||lname from emp;

OUTPUT:- 'Mr. '||fname||' '||lname

Mr. Arun Purun
Mr. Tarun Arun
Mr. Sirun Kirun
Mr. Nutan Purun

* || is supported by Oracle
* **|| is not supported by MySQL**

concat (str1, str2)

select concat(fname,lname) from emp;

OUTPUT: -
ArunPurun
TarunArun
SirunKirun
NutanPurun

select concat(concat(fname,' '),lname) from emp; -> (function within function)

* max upto 255 levels for a function within function

UPPER case: -

select upper(fname) from emp; -> (only displays)

OUTPUT: -
ARUN
TARUN
SIRUN
NUTAN

update emp set fname = upper(fname); -> (updates in table)

Solution for case-insensitive query in Oracle: -

select * from emp where upper(fname) = 'ARUN';

select * from emp where lower(fname) = 'arun';

INITCAP Initial Capital:- (First letter capital)

select initcap (ename) from emp; -> supported by Oracle (**not supported by MySQL**)

OUTPUT: -

Arun
Tarun
Sirun
Nutan

select concat(upper(substr(fname,1,1)),lower(substr(fname,2))) from emp;

EMP Table
ENAME

Arun Purun
Tarun Arun
Sirun Kirun
Nutan Purun

LPAD: - (Right justification puts blank spaces at the left hand side)

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

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

USES:-
a. Right justification
b. cheque printing

RPAD: -

```
select rpad(ename,25,' ') from emp;

select rpad(ename,25,'*') from emp;
```

USES: -
a. Left justification of numeric data
b. to convert varchar to char
c. Centre-justification (use cobo of lpad & rpad)

LTRIM: - (removes black spaces on left hand side)

```
select ltrim(ename) from emp;
```

USES: -
a. Left justification

RTRIM: - (removes black spaces on right hand side)

```
select rtrim(ename) from emp;
```

USES: -
a. Right justification of char data lpad(rtrim(ename),...)
b. to convert char to varchar

TRIM: - (removes black spaces from both the sides)

select trim(ename) from emp;

SUBSTR: - (displays from the given position)

select substr(ename,3) from emp; -> (3 is starting position)

select substr(ename,3,2) from emp; -> (3 is starting position,2 is number of characters(gets 3rd & 4th letter))

select substr(ename,-3,2) from emp; -> (-3 is starting position, it will start from right side, we will get last 3 letters of the string)

USES: -
a. used to extract a part of string

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

substr('New Mumbai',5) ; -> Mumbai

REPLACE: - (replaces the string)

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

select replace(ename,'un','xyz') from emp; un->xyz

select replace(ename,**'un','xyz'**) from emp; -> **will not work in MySQL 3rd parameter compulsory in MySQL** (works in Oracle)

USES: -
a. Encoding and Decoding
b. Encryption and Decryption
c. Masking of ATM
d. Card Number

TRANSLATE: -

select translate(ename,'un','xy') from emp;
u -> x
n -> y

select translate(ename,'un','xyz') from emp;
u -> x
n -> y

-> z

select translate(ename,'un','x') from emp;
u -> x
n ->
* **TRANSLATE function is not available in MySQL** (available in Oracle)

INSTR: - (returns starting position of string)

select instr(ename,'un') from emp; -> returns starting position of string

USES: -
a. used to check if one string exists in another string

select instr(ename,'un',4) from emp;
4 -> starting position from where it will start searching

select instr(ename,'un',4,2) from emp;
4 -> starting position from where it will start searching
2 ->return position only when un is repeated twice (2nd occurence)

select instr(ename,'un',-4) from emp;
4 -> starting position from last 4th, it will start searching

* **INSTR is available in MySQL but 3rd and 4th parameter not allowed in MySQL**

LENGTH: - (returns the length of string)

select length(ename) from emp;
* for varchar as char has fixed lenght

ASCII: -(returns the ascii value of 1st letter)

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

select ascii ('z') from dual;

* DUAL is a system table
* it contains only 1 row and column
* DUAL is a dummy table (present in all RDBMS)

select substr('New Mumbai', 1,3) from dual;
select 'Welcome to CDAC Mumbai' from dual;
select 10+10 from dual;

CHAR:- (returns the character corresponding to ascii value)

In MySQL: -
select char (65 using utf8) from dual; -> **A**

-->> where utf8 is the given character set for US English else default binary character set

In Oracle:

select chr (65) from dual; -> A

SOUNDEX: -
(removes the vowels from both string and then compares) (a, e, i, o, u, y -> US)
select * from emp where soundex(ename) = soundex('Aroon');

DAY 5

Number Functions: -

Sal

1234.567
1561.019
1375.516
1749.167

In MySQL: -

sal float
select round(sal) from emp;
select round(sal,1) from emp; -> round off the sal till 1 decimal place
select round(sal,2) from emp; -> round off the sal till 2 decimal place
select round(sal,-2) from emp; -> round off the sal on left side till 2 decimal place

In Oracle: -
sal number (7,3)
1234.567

TRUNCATE: - (removes the decimal point numbers)

In MySQL: -
select truncate(sal,0) from emp;

select truncate(sal,1) from emp;

select truncate(sal,2) from emp;

select truncate(sal,-2) from emp;

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 Ceiling: - (adds 1 to the last no by removing decimal point)

select ceil(sal) from emp;

FLOOR: - (removes decimal and goes for lower no)

select floor(sal) from emp;

select truncate (3.6,0), floor (3.6), truncate (-3.6,0), floor (-3.6) from dual;

3 3 -3 -4

SIGN: -

-1
0
1

select sign (-15) from dual; -> **-1**

Uses: -
1. check if num is +ve or -v
2. sign(SP-CP)
3. sign(temperature)
4. sign(blood_group)
5. sign(medical_report)
6. sign(bank_balance)
7. sign(sensex)

MOD: -

select mod(9,5) from dual; -> 4

select mod(8.22,2.2) from dual; -> 1.62

SQRT: -

select sqrt(81) from dual; -> 9

POWER: -

select power(10,3) from dual; -> 1000

select power(10,1/3) from dual; -> 10*0.33 =

** does not work in SQL

** works in Oracle PL/SQL programs
in SQL, if you want to perform exponentiation, then you will have to use the POWER function

ABS: -

select abs(-10) from dual;-> 10

x -> radians

sin(x)
cos(x)
tan(x)
sinh(x) -> not supported by MySQL (works in Oracle)
cosh(x) -> not supported by MySQL (works in Oracle)
tanh(x) -> not supported by MySQL (works in Oracle)

ln(y)
log(n,m)

Date and Time Functions: -

Date (1st Jan 1000 AD to 31st Dec 9999 AD)
Time
Datetime
Year

* internally date is stored as a fixed-length number and it occupies **7 Bytes** of storage

datel-date2 -> returns number of days between the 2 dates

select sysdate() from dual; -> return date and time when the statement executed

sysdate -> return DB server date and time

select now() from dual; -> return date and time when the statement began to execute

select sysdate(), now() from dual;

sysdate() -> used for date, time, clock display

now() -> used to maintain logs of operations, e.g. maintains logs of DML operations

select adddate(sysdate(),1) from dual; -> shows date of tommorow

select adddate(sysdate(),-1) from dual; -> shows date of yesterday

select datediff(sysdate(),hiredate) from dual; -> returns no of days between 2 dates

select date_add(hiredate,interval 2 month) from dual; -> adds 2 months to the date

select date_add(hiredate,interval -2 month) from dual; -> substracts 2 months to the date

select date_add(hiredate,interval 1 year) from dual; -> adds 1 year to the date

select last_day(hiredate) from dual; -> returns last date of month

select dayname(sysdate()) from dual; -> returns day of the date

select addtime('2020-01-10 11:00:00',1') from dual; -> adds 1 second to time

select addtime('2020-01-10 11:00:00',01:30:45') from dual; -> adds 01:30:45 to time

LIST Functions (independent of datatype)

EMP

ename	sal	comm
A	5000	500
B	6000	null
C	null	700

select * from emp where comm = null; -> returns null

select * from emp where comm != null; -> returns Not null

* any comparison done with null, returns null

PESSIMISTIC Querying:- searching for null values

IS NULL: - (Special Operator)

select * from emp where comm is null;

select * from emp where comm is not null;

****0 is not null

select sal+comm from emp;

* any operation done with null, returns null

OUTPUT: -
5500
null
null

IFNULL: - (In MySQL)

select sal + ifnull(comm,0) from emp; -> if comm is null return 0, else return comm

OUTPUT: -
5500

6000
null

select ifnull(sal,0) + ifnull(comm,0) from emp; ->if sal is null return 0, else return sal, if comm is null return 0, else return comm

OUTPUT: -
5500
6000
700

ifnull(comm,0)
ifnull(comm,100)
ifnull(city,'Goa')
ifnull(orderdate,'2021-04-01')

NVL: - (In Oracle)

nvl(comm,0)
nvl(comm,100)
nvl(city,'Goa')
nvl(orderdate,'01-APR-2021')

GREATEST Function: - (compares returns greatest among values)

EMP

ename	sal	deptno
A	1000	10
B	2000	10
C	3000	20
D	4000	30
E	5000	40

select greatest(sal,3000) from emp;

OUTPUT: -
3000
3000
3000
4000
5000

* used to set a lower limit on some value
e.g. bonus = 10% of sal, min Rs. 300 guaranteed

select greatest(sal*0.1,300) "BONUS" from emp;

greatest(val1,val2,val3,.....,val255) -> upto 255 values
greatest('date1','date2','date3')

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

LEAST Function: - (compares returns smallest among values)

select least(sal,3000) from emp;

OUTPUT: -
1000
2000
3000
3000
3000

* used to set an upper limit on some value
e.g. cashback = 10% of amt, max cashback = Rs. 10000
select least(amt*0.1,300) "CASHBACK" from ORDERS;

least(val1,val2,val3,.....,val255) -> upto 255 values
least('str1','str2','str3','str4')
least('date1','date2','date3')
set x = least(a,b,c,d);

CASE expression: -

```
select
case
when deptno = 10 then 'Training'
when deptno = 20 then 'Exports'
when deptno = 30 then 'Sales'
else 'Others'
end "DEPTNAME"
from emp;
```

OUTPUT: -

deptno	DEPTNAME
10	Training
10	Training
20	Exports
30	Sales
40	Others

* if you don't supply ELSE and if some undefined value is present in the table, then it returns a null value

```
select
case
when deptno = 10 then 'Ten'
when deptno = 20 then 'Twenty'
when deptno = 30 then 'Thirty'
when deptno = 40 then 'Forty'
end "DEPTCODE"
from emp;
```

OUTPUT: -

deptno	DEPTCODE
10	Ten
10	Ten
20	Twenty
30	Thirty
40	Forty

if sal < 3000 then REMARK = 'Low Income'
if sal = 3000 then REMARK = 'Middle Income'
if sal > 3000 then REMARK = 'High Income'

```
select
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;
```

```
select user() from dual;
```
-> **IN MySQL**

```
select user from dual;
```
-> In Oracle

In MySQL: -

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	null
5	Thomas	8000	2	C	4

Single-Row Functions:-

* will operate on 1 row at a time
* Character, Number, Date, List, Environment Functions e.g. upper (ename), round (sal), etc.

Multi-Row Functions: -

* will operate on multiple rows at a time
* Group Functions
  e.g. sum (sal), etc.

SUM: -

select sum(sal) from emp; -> 35000

Assumption, last row SAL is null: -

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	null
5	Thomas	null	2	C	4

select sum(sal) from emp; -> 27000

* **null values are not counted by group functions**

AVG: -

select avg(sal) from emp -> 27500/4 = 6750

select avg(ifnull(sal,0)) from emp -> 27500/5 = 5400

MIN: -

select min(sal) from emp; -> 3000

select min(ifnull(sal,0)) from emp; -> 0

MAX: -

select max(sal) from emp; -> 9000
select max(sal)/min(sal) from emp; -> 9000/3000 = 3

COUNT: -

select count(sal) from emp; -> 4 returns a COUNT of number of rows where sal is not having a null value

select count(*) from emp; -> 5 returns a COUNT of total number of rows in the table

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

select sum(sal)/count(*) from emp; -> 27000/5(FASTER)

select avg(ifnull(sal,0)) from emp; -> (SLOWER)

Assumption, last row SAL is 8000: -

select sum(sal) from emp where deptno = 1; -> 18000

* 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

select avg(sal) from emp where job = 'C'; -> 6000

COUNT Query: - (counting the numbers of query hits)

select count(*) from emp where sal > 7000; -> 3

sum(column)
avg(column)
min(column) min(ename),min(hiredata)
max(column) max(ename),max(hiredata)
count(column) count(ename),count(hiredata)
count(*)
stddev(column)
variance(column)

When you install, 3 users are automatically created:

scott/tiger
* regular user having connect, resource, create view privileges
* this user can be dropped

drop user scott;

system/manager
* DBA privileges (similar to root user of MySQL)
* this user can be dropped

sys/change_on_install
* owner of database
* owner of system tables
* this user cannot be dropped
* most important user

Run SQL command line

SQL> connect
SQL> create user <username> identified by <password>;
SQL> grant connect, resource, create view to <username>;
SQL> select * from all_users; -> shows users
SQL> select *

DAY 6

Group Functions

SUMMARY REPORT: -

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

* YOU CANNOT SELECT A REGULAR COLUMN WITH A GROUP FUNCTION

select ename,min(sal) from emp; -> ERROR in Oracle
(works in MySQL but output is meaningless)

select count(ename),min(sal) from emp;

* YOU CANNOT SELECT A SINGLE ROW FUNCTION WITH A GROUP FUNCTION

select upper(ename),min(sal) from emp; -> ERROR in Oracle
(works in MySQL but output is meaningless)

* YOU CANNOT USE GROUP FUNCTION IN THE WHERE CLAUSE

select * from emp where sal > avg(sal);

GROUP BY clause: - (used for grouping)

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	null
5	Thomas	8000	2	C	4

select sum(sal) from emp where deptno = 1;

sum(sal) deptwise: -

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

SELECT clause -> select deptno, sum(sal)
FROM clause -> from emp
GROUP BY clause -> group by deptno;

OUTPUT: -

deptno	sum(sal)
1	18000
2	17000

1. rows retrieved from DB server Hd to server RAM (WHERE clause is used to retrieve the rows from DB server HD to server RAm)
2. sorting dept wise
3. grouping dept wise
4. summation dept wise
5. HAVING clause
6. ORDER BY clause

select sum(sal) from emp group by deptno;

OUTPUT: - sum(sal)

18000
17000

* whichever column is present in GROUP BY clause, it may or may not be present in SELECT clause

select deptno, max(sal) from emp group by deptno;

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

* WHERE clause is used to retrieve the rows from DB server HD to server RAM
* WHERE clause has to be specified before GROUP BY clause

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

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

* the position of columns in SELECT clause and the position of column in GROUP BY clause need not be same
* the position of columns in SELECT clause will determine the position of columns in the output
* the position of columns in GROUP BY clause will determine the sorting order , grouping order, summation order and hence the speed of processing
* no upper limit on the munber of columns in GROUP BY clause

select ..... group by country,state,district,city; -> **FASTER**

select ..... group by city,district,state,country; -> SLOWER

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

HAVING clause: -

select deptno, sum(sal) from emp group by deptno having sum(sal) > 17000; -> its recommended that only group functions should be used in HAVING clause

OUTPUT: -
deptno sum(sal)
------ --------
1 18000

* HAVING clause works after the summation takes place

select deptno, sum(sal) from emp group by deptno having sum(sal) > 7000; -> ERROR

* 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 takes place
* whichever column is present in SELECT clause, it can be used in HAVING clause

select deptno, sum(sal) from emp group by deptno having dept no = 1; -> will work but it is inefficient

OUTPUT: -

deptno sum(sal)
1 18000

select deptno, sum(sal) from emp group by deptno having sum(sal) > 17000 and sum(sal) < 25000;

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

* in the HAVING clause you may use a group function that is not present in SELECT clause

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

OUTPUT: -

deptno sum(sal)
1 18000
2 17000

* ORDER BY clause is the last clause in SELECT statement

select deptno, sum(sal) from emp group by deptno order by 2;

select.....from.....where.....group by.....having.....order by.....;

select deptno, sum(sal) from emp where sal > 7000 group by deptno having sum(sal) > 10000 order by 1;

In Oracle: -

select max(sum(sal)) from emp group by deptno; -> nesting of GROUP Functions is allowed in Oracle RDBMS (Not supported in any other RDBMS)

OUTPUT: - max(sum(sal))

18000

In MySQL: -

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

OUTPUT: -

max(sum_sal)
18000

MATRIX Report: -

select deptno, count(*), min(sal), max(sal), sum(sal) from emp gorup by deptno order by 1;

JOINS: - (V. IMP)

* to view/combine the columns of 2 or more tables

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	null
5	Thomas	8000	2	C	4

DEPT

deptno	dname	location
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

DATA REDUNDACY - unnecessary duplication of data (wastage of HD space)

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

tablename.columnname

dept -> driving table
emp -> driven table
* In order for the join to work faster, preferably the driving table should be table with lesser number of rows

OUTPUT: -

ename	dname
Arun	TRN
Ali	TRN
Kirun	TRN
Jack	EXP
Thomas	EXP

* the common column in both the tables, the column name need not to be same in both the tables, because the same column may have a different meaning in the other table
* what matters is the datatype of the column has to match in both the tables, and there has to be some sensible relation on whose basis you are writing the join

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

```
select dname, ename from emp, dept where dept.deptno = emp.deptno order by 1;
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;
```

select deptno, dname, loc, ename, job, sal from emp, dept where dept.deptno = emp.deptno order by 1; -> ERROR: column ambiguity defined

select dept.deptno, dname, loc, ename, job, sal from emp, dept where dept.deptno = emp.deptno order by 1;

select dept.deptno, dept.dname, dept.loc, emp.ename, emp.job, emp.sal from emp, dept where dept.deptno = emp.deptno order by 1; -> GOOD PROGRAMMING PRACTICE

select upper(dname) as dname, sum(sal) from emp,dept where dept.deptno = emp.deptno group by upper(dname) having..... order by.....;

OUTPUT: -

dname	sum(sal)
TRN	18000
EXP	17000

Types of Joins: -

1. EQUIJOIN (also known as NATURAL JOIN)

* join based on equality join(condition)
* shows matching rows of both the tables
* data is not stored in one table; data is stored in multiple tables;if you want to view/combine the columns of 2 or more tables then you will write Equijoin
* most frequently used join (more tahn 90%) hence it is also known as NATURAL JOIN

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

dept -> driving table
emp -> driven table

OUTPUT: -

dname	ename
TRN	Arun
TRN	Ali
TRN	Kirun
EXP	Jack
EXP	Thomas

2. INEQUIJOIN (also known as NON-EQUIJOIN)

* join based on inequality condition
* shows non-matching rows of both the tables
* used in Exception Reports

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

OUTPUT: -

dname	ename
TRN	Jack
TRN	Thomas
EXP	Arun
EXP	Ali
EXP	Kirun
MKTG	Arun
MKTG	Ali
MKTG	Kirun
MKTG	Jack
MKTG	Thomas

3. OUTER JOIN

* join with (+) sign (supported only in Oracle RDBMS & not supported by any other RDBMS)
* shows matching rows of both the tables
  plus
  the non-matching rows of "OUTER" table
* **Outer table** -> table which is on Outer/Opposite side of = sign
* used in Master-Detail Report (Parent-Child Report)

a. Half Outerjoin

* one of the loop is Do-While loop and one is for loop

1. Right Outerjoin
2. Left Outerjoin
3. Full Outerjoin

* (+) sign on both the sides (theoretically)
* shows matching rows of both the tables
  plus
  the non-matching rows of both the table
* based on nested Do-While loop

select dname, ename from emp, dept where dept.deptno = emp.deptno (+) ; -> Right Outerjoin

dept (outer loop) (Do-While loop)
emp (inner loop) (For loop)

OUTPUT: -

dname	ename
TRN	Arun
TRN	Ali
TRN	Kirun
EXP	Jack
EXP	Thomas
MKTG	null

select dname, ename from emp, dept where dept.deptno (+) = emp.deptno; -> Left Outerjoin

dept (outer loop) (For loop)
emp (inner loop) (Do-While loop)

*** Suppose the table has 6th row as follows

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	null
5	Thomas	8000	2	C	4
6	Scott	6000	99		

DEPT

deptno	dname	location
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

OUTPUT: -

dname	ename
TRN	Arun
TRN	Ali
TRN	Kirun
EXP	Jack
EXP	Thomas
null	Scott

select dname, ename from emp, dept
where dept.deptno = emp.deptno (+)
union -> Full OuterJoin
select dname, ename from emp, dept
where dept.deptno (+) = emp.deptno;

OUTPUT: -

dname	ename
TRN	Arun
TRN	Ali
TRN	Kirun
EXP	Jack
EXP	Thomas
MKTG	null
null	Scott

ANSI syntax for RIGHT Outerjoin: - (supported by all RDBMS including MySQL & 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 & Oracle)

```
select dname, ename from emp left outer join dept
on (dept.deptno = emp.deptno);
```

ANSI syntax for FULL Outerjoin: - (supported by all RDBMS except MySQL)

```
select dname, ename from emp full outer join dept
on (dept.deptno = emp.deptno);
```

To achieve full outer join in MySQL:-

* you will have to take UNION of ANSI syntax for RIGHT Outerjoin and ANSI syntax for LEFT Outerjoin

```
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);
```

INNER Join: - *****do not mention in interviews unless explicitly asked by interviewer (jyada shanpatti nahi krneka)

* by default every join is INNER join , putting a (+) sign is what makes it an Outerjoin

Day 7

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
* FASTEST join because you don't have a WHERE clause, and therefore no seraching is involved

select dname, ename from emp, dept; -> **FASTER**

select ename, dname from dept, emp; -> SLOWER

dept -> driving table
emp -> driven table

OUTPUT:-

dname	ename
TRN	Arun
TRN	Ali
TRN	Kirun
TRN	Jack
TRN	Thomas
EXP	Arun
EXP	Ali
EXP	Kirun
EXP	Jack
EXP	Thomas
MKTG	Arun
MKTG	Ali
MKTG	Kirun
MKTG	Jack
MKTG	Thomas

USES: -

* used for printing purposes,

e.g. in the University, in STUDENTS table you have all the students names, in SUBJECTS table you have all the subjects names;when you are printing the marksheet for the students, then every student name is combined with each and every subject name, you will require a CARTESIAN JOIN

5. SELF JOIN

* joining a table to itself
* used when parent and child column both are present in same table
* based on Recursion
* this is SLOWEST join

```
select a.ename, b.ename from emp as b, emp as a
where a.mgr = b.empno;
```

OUTPUT:-

a.ename	b.ename
Arun	Jack
Ali	Arun
Kirun	Arun
Thomas	Jack

Joining 3 or more tables: -

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	null
5	Thomas	8000	2	C	4

DEPT

deptno	dname	location
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

DEPTHEAD

deptno	dhead
1	Arun
2	Jack

(5) (3) (2)

```
select dname, ename, dhead from emp, dept, depthead
where depthead.deptno = dept.deptno
and dept.deptno = emp.deptno;
```

OUTPUT:-

dname	ename	dhead
TRN	Arun	Arun
TRN	Ali	Arun
TRN	Kirun	Arun
EXP	Jack	Jack
EXP	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 (Emp : Projects) or (Projects : 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	null
5	Thomas	8000	2	C	4

PROJECTS

pno	pname	clientname
P1	CGS	Deloitte
P2	AMS	Morgan Stanley
P3	PPS	ICICI Bank
P4	Macro Dev	BNP Parivas
P5	Website Dev	AMFI

PROJECTS_EMP -> INTERSECTION Table

pno	empno
P1	1
P1	2
P1	4
P2	1
P2	3
P3	2
P3	4
P3	5

* **INTERSECTION table is required for Mnay : Many Relationship**

```
select pname, clientname, ename from projects_emp, emp, projects
where project_emp.pno = projects.pno
and projects_emp.empno = emp.empno;
```

Sub - Queries: - (V. Imp)

(Nested Queries) (Query within query) (SELECT within SELECT)

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	null
5	Thomas	8000	2	C	4

Display the ENAME who is receiving min(sal): -

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

OUTPUT: - Kirun

```
select ename from emp
where sal = (select min(sal) from emp
where deptno = (select.....));
```

* max upto 255 levels for sub-queries
* **JOIN is FASTER than SUB-QUERY** (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);
```

Display all the rows with same deptno as 'Thomas': -

```
select * from emp where deptno =
(select deptno from emp where ename = 'Thomas');
```

Display all the rows with same job as 'Kirun': -

```
select * from emp where job =
(select job from emp where ename ='Kirun');
```

Using sub-queries with DML commands: -

In Oracle: -

```
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');
```

In MySQL: -

* you cannot UPDATE or DELETE from a table from which you are currently SELECTing

Solution: -

```
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 tempp.job from (select job from emp where ename ='Kirun') as tempp);
```

Multi-row sub-queries: -
(sub-query returns multiple rows): -

Display all the rows who are receving the sal equal to any one of managers: -

select * from emp where sal =
any (select sal from emp where job = 'M'); -> **Recommended**

```
select * from emp where sal in
(select sal from emp where job = 'M');
```

```
select * from emp where sal >=
(select min(sal) from emp where job = 'M');
```

To make it work faster: -

1. Try to solve the problem using join instead of sub-query because using a join you solve the problem using one SELECT statement whereas using sub queries you solve the problem using two or more SELECT statements; the more the number of SELECT statements, the slower it will be

2. Try 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: -

EMP

empno	ename	sal	deptno	job	mgr
1	Arun	8000	1	M	4
2	Ali	7000	1	C	1
3	Kirun	13000	1	C	1
4	Jack	9000	2	M	null
5	Thomas	8000	2	C	4

Display the rows who are receiving a sal greater than all of the Managers: -

```
select * from emp where sal > all
(select sal from emp where job ="M");
```

ANY -> Logical OR
IN -> Logical OR
ALL -> Logical AND

```
select * from emp where sal >
(select max(sal) from emp where job ='M");
```

Assumption, 3rd row sal is 3000: -

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	null
5	Thomas	8000	2	C	4

Using sub-query in the HAVING clause: -

Display the DNAME that is having max(sum(sal)): -

In Oracle: -

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

OUTPUT: -

deptno	sum(sal)
1	18000
2	17000

select sum(sal) from emp group by deptno;

OUTPUT: -

sum(sal)
18000
17000

select max(sum(sal)) from emp group by deptno;

OUTPUT: -

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);

OUTPUT: -

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);

OUTPUT: -

dname	sum(sal)
TRN	18000

In MySQL: -

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

OUTPUT: -

deptno	sum(sal)
1	18000
2	17000

select sum(sal) from emp group by deptno;

OUTPUT: -

sum(sal)
18000
17000

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

OUTPUT: -

max(sum_sal)
18000

```
select deptno,sum(sal) from emp group by deptno
having sum(sal) = (select max(sum_sal) from
(select sum(sal) as sum_sal from emp group by deptno) as tempp;
```

OUTPUT: -

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) as sum_sal from emp group by deptno) as tempp;
```

OUTPUT: -

dname	sum(sal)
TRN	18000

DAY 8

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	null
5	Thomas	8000	2	C	4

DEPT

deptno	dname	location
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

Correlated Sub-Query: - (using EXISTS operator)

* this is the exception when sub-query is faster than join

Display the DNAME that the employees belong to: -
Solution 1:-

```
select deptno from emp;
```

OUTPUT: -

```
deptno
------
1
1
1
2
2
```

```
select distinct deptno from emp;
```

OUTPUT: -

```
deptno
------
1
2
```

```
select dname from dept where deptno = any
(select distinct deptno from emp);
```

OUTPUT: -

```
dname
------
TRN
EXP
```

```
select dname from dept where deptno in
(select distinct deptno from emp);
```

OUTPUT: -

dname

TRN
EXP

```
select dname from dept where deptno not in
(select distinct deptno from emp);
```

OUTPUT: -

dname

MKTG

Solution 2: -

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

OUTPUT: -

dname

TRN
TRN
TRN
EXP
EXP

```
select distinct dname from emp, dept
where dept.deptno = emp.deptno;
```

OUTPUT: -

dname

TRN
EX

Solution 3: -

* Whenever you have a join, along with DISTINCT, to make it work faster, use correlated sub-query (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);
```

OUTPUT: -

dname

TRN
EXP

* 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 values back to main query
* if sub-query returns a TRUE value, then main query is executed for that row
* if sub-query returns a FALSE value, then main query is not executed for that row
* unlike earlier we do not use DISTINCT, hence no sorting taakes place at server RAM, this speeds it up
* unlike a traditional join, the number of full tables scans is reduced, this further speeds it up

NOT EXISTS: -

```
select dname from dept where not exists
(select deptno from emp
where dept.deptno = emp.deptno);
```

OUTPUT: -

```
dname
------
MKTG
```

SET Operators:-

* based on SET theory

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;
```

OUTPUT: -

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

union -> will combine the output of both the SELECTs and it will supress the duplicates

```
select empno1, ename from emp1
        union
select empno2, ename from emp2 order by 1;
```

OUTPUT: -

empno1	ename
1	A
2	B
3	C
4	D
5	E

```
select empno1, ename from emp1
        union all
select empno2, ename from emp2 order by 1;
```

OUTPUT: -

empno1	ename
1	A
1	A
2	B
2	B
3	C
4	D
5	E

union all -> will combine the output of both the SELECTs and the duplicates are not supressed

INTERSECT: -

```
select empno1, ename from emp1
        intersect
select empno2, ename from emp2 order by 1;
```

OUTPUT: -

empno1	ename
1	A
2	B

intersect -> will return what is common in both the SELECTs and it will supress the duplicates

MINUS: -

```
select empno1, ename from emp1
        minus
select empno2, ename from emp2 order by 1;
```

OUTPUT:

empno1	ename
3	C

minus -> will return what is present in first SELECT and what is present in second SELECT and the duplicates are suppressed

* max upto 255 SELECTS
* execution is top to bottom

```
select .....
        union
select .....
        minus
select .....
        union
select .....
        union all
select .....
        intersect
select .....
        order by x;
---------------------------------------

select .....
        union
(select .....
        minus
select .....)
        union
(select .....
        union all
select .....)
        intersect
select .....
        order by x;
```

* multiple SELECTs, brackets for nesting -> not supported by MySQL
* **union, union all are supported by all RDBMS**
* intersect, minus are supported by Oracle, not supported by MySQL

PSEUDO Columns: -

* fake columns (virtual columns)
* not a column of the table, but you can use it in SELECT statement
  e.g. computed columns (ANNUAL = sal*12), expressions (NET_EARNINGS = sal+comm), function-based columns (TOTAL = sum(sal))

RDBMS supplied Pseudo columns: -

```
select ename, sal from emp;

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

ROWNUM -> returns the row number

select rownum, ename, sal from emp where rownum = 1;

select rownum, ename, sal from emp where rownum < 4;

select rownum, ename, sal from emp where rownum = 4;

select rownum, ename, sal from emp where rownum > 4;

select rownum, ename, sal from emp order by ename;

select rowname, ename, sal from
(select ename, sal from emp order by ename);

INLINE VIEW -> if you use sub-query in the FROM clause, it is known as INLINE VIEW

select rowid, ename,sal from emp;

ROWID: -

* it is a row address of the row in the DB server HD
* (actual physical memory location where that row is stored)
* fixed length encrypted string of 18 characters
* when you select from atable, the order of rows in the output will be in ascending order of row address
* when you SELECT from atable, the order of rows in the output will be in ascending order of ROWID
* No two rows of any table in the entire DB can have same ROWID
* ROWID works as unique identifier for every row in the DB
* When you UPDATE a row the ROWID may change
* You can use ROWID to UPDATE or DELETE the duplicate rows

ROWID is used internally by the RDBMS: -
1. To distinguish between 2 rows in the DB
2. For row locking
3. To manage the INDEXEs
4. To manage the CURSORS
5. Row management

* ROWID is present in Oracle and you can view it
* **ROWID is present in MySQL and you can NOT view it**
* ROWNUM is present in Oracle and you can view it
* **ROWNUM is not present in MySQL**

ALTER table: - (DDL command)

EMP

empno	ename	sal
101	Scott	5000
102	King	6000

* rename a table
* add, drop a column
* increase width of column

INDIRECTLY: -

* reduce width of column
* change datatype of column
* copy rows from one table into another table
* copy a table
* copy structure of table
* rename a column
* change position of columns in table structure
(because of null values, for storage conditions)

RENAME a Table: - (DDL command)

rename table emp to employees; -> **In MySQL**

rename emp to employees; -> In Oracle

ADD a column: -

```
alter table emp add gst float;
```

DROP a column: -

```
alter table emp drop column gst;
```

INCREASE WIDTH of column: -

In MySQL: -

```
alter table emp modify ename varchar(30);
```
-> data will get truncated

In Oracle: -

```
alter table emp modify ename varchar2(30);
```

* you can reduce the width provided the contents are null

```
alter table emp add x varchar2(25);
update emp set x = ename, ename = null;
alter table emp modify ename varchar2(20);
/* Data testing with x column */
update emp set ename = x;
alter table emp drop column x;
```

CHANGE DATATYPE of column

In Oracle: -

* you can change the datatype provided the contents are null

update emp set empno = null;
alter table emp modify empno char(4);

copy rows from one table into another table: -

insert into emp select * from emp2;

to copy specific rows only: -

insert into emp select * from emp2 where.....;

copy a table: -

create table emp_copy as select * from emp;

copy 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 possible	ROLLBACK not possible
*can use WHERE clause	cannot use WHERE caluse with TRUNCATE
*Free space is not deallocated	Free space is deallocated
*when you delete the rows delete triggers on table will execute	when you truncate a table delete tables on triggers will not execute

Method 3: -

create table emp_struct as
select * from emp where 1 = 2; -> give an impossible where caluse so that no row will get copied

rename a column: -

rename table emp_copy to emp; -> **In MySQL**

rename emp_copy to emp; -> In Oracle

change position of columns in table structure: -

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

Privileges: -

GRANT / REVOKE (DCL commands)

create users scott,cdac,aaba,etc.

GRANT: -

SCOTT_MYSQL> grant select on emp to king;

SCOTT_MYSQL> grant insert on emp to king;

SCOTT_MYSQL> grant update on emp to king;

SCOTT_MYSQL> grant delete on emp to king;

SCOTT_MYSQL> grant select, insert on emp to king;

SCOTT_MYSQL> grant all on emp to king;

SCOTT_MYSQL> grant select on emp to king, cdac;

SCOTT_MYSQL> grant select on emp to public; -> public means all users

REVOKE: -

SCOTT_MYSQL> revoke select on emp to king;

to see the permissions granted and received:-

******SCHEMA IS A SYNONYM FOR DATABASE**

select * from information_schema.table_privileges; -> In MySQL

KING_MYSQL> select * from cdac.emp;

cdac -> schema/database name
emp -> table name

KING_MYSQL> insert into cdac.emp values ......;

KING_MYSQL> update cdac.emp set ......;

KING_MYSQL> delete from cdac.emp ......;

SCOTT_MYSQL> grant select, insert on emp to king with grant option;

KING_MYSQL> grant select on cdac.emp to aaba;

DAY 9

INDEXES: -

Types of Indexes: -

1. Normal index **(MySQL)**
2. Unique index
3. Clustered index
4. Bitmap index 3 to 6 Advanced (Oracle)
5. Index-Organized table
6. Index partitioning

NORMAL INDEX: -

* present in all RDBMS, all DBMS, and some programming languages also
* to speed up the search operations (for faster access)
* to speed up SELECT statement with a WHERE clause
* indexes are automatically invoked by MySQL as and when required
* indexes are automatically updated by MySQL for all the DML operations
* duplicate values are stored in index
* null values are not stored in an index
* no upper limit on the number of indexes per table
* larger the number of indexes, the slower wold be the DML operations
* cannot index TEXT and BLOB columns
* if you have multiple INDEPENDENT columns in the WHERE clause, then you should create seperate indexex for each column, MySQL will use the necessary indexes as and when required

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

In Other RDBMS: -

select * from emp where empno = 1;

IND_EMPNO

rowid	empno
X003	1
X004	2
X005	3
X002	4
X001	5

use index ind_empno;

select * from emp where empno is null; -> SLOWER

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

IND_ENAME

rowid	ename
X001	A
X002	A
X003	C
X004	D
X005	E

select * from emp where ename = 'C';

IND_SAL

rowid	sal
X001	5000
X002	6000
X003	7000
X005	8000
X004	9000

select * from emp where sal > 7000;

select * from emp where empno = 2;

select * from emp where sal > 5000;

select * from emp where empno = 2 and sal > 5000;

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

IND_DEPTNO_EMPNO

rowid	deptno	empno
X001	1	1
X002	1	2
X003	1	3
X004	2	1
X005	2	2

DEPTNO -> **PRIMARY INDEX KEY**

EMPNO -> **SECONDARY INDEX KEY**

```
select * from emp where deptno = 1 and empno = 1;
```

COMPOSITE INDEX -> to combine two or more INTET-DEPENDENT columns in a single index, also known as a COMPLEX INDEX

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**

1. Read
2. Compile
3. Plan
4. Execute

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

Conditions when an index should be created: -

1. If SELECT statement retrieves < 25% of table data
2. PRIMARY KEY columns and UNIQUE columns should always be indexed
3. Common columns in join operations should always be indexed

IND_EMPNO

rowid	empno
X001	1
X002	2
X003	3
X004	4
X005	5

```
select * from emp where empno = 1;

select * from emp where empno = 5;

select * from emp where empno < 2;
```

select * from emp where empno > 1; -> **MySQL will use the index but it will be very slow**

DEPT

rowid	deptno	dname	location
Y011	1	TRN	Bby
Y012	2	EXP	Dlh
Y013	3	MKTG	Cal

I2

rowid	deptno
X001	1
X002	1
X003	1
X004	2
X005	2

I1

rowid	deptno
Y011	1
Y012	2
Y013	3

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

Syntax to create INDEX: - **(DDL command)**

```
create index indexname on table(columnname);
```

```
create index indexname on table(column1,column2);
```
-> **composite index**

* no upper limit on creating indexes on a table in MySQL and Oracle
(**banake chod deneka RAM bharose**)

```
create index i_emp_empno on emp(empno);
```

i_emp_empno

rowid	empno
X001	1
X002	2
X003	3
X004	4
X005	5

```
select * from emp where empno = 1;
```
-> Execute very fast (makkhan ke mafik)

```
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); -> **Descending**

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

TO DROP INDEX: -

IN MySQL: -
drop index i_emp_empno on emp;

IN Oracle: -
drop index i_emp_empno;

create index i_orders_onum on emp(onum desc); -> latest (new) orders will stored first at the top, older orders would be below

to see which all indexes are created for specific table: -

show indexes from table;

show indexes from emp;

to see all indexes on all table in the DB: -
use information_schema;
select * from statistics;

create table emp_copy as select * from emp;

* if you create a table using sub-query, then indexes created on original table will not be copied into the new table, if you want then you have to create them manually

UNIQUE INDEX: -

create unique index i_emp_empno on emp(empno);

* works like a normal index, but it performs one extra function, it will not allow you to INSERT duplicate values for empno

* **Oracle & MySQl doesn't allow** more than one indexes on same column

EMP

empno	ename	sal	deptno
1	A	5000	1
2	A	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

CONSTRAINTS: - (V. IMP)

* limitations/restrictions imposed on a table

PRIMARY KEY (Primary column): -

* column or set of columns that uniquely identifies a row
* duplicate values are not allowed (**has to be unique**)
* null values are not allowed (**it's a mandatory column**)
* it's recommended tata every table should have a Primary Key
* purpose of Primary Key is row uniqueness (with the help of Primary Key column, 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 INTER-DEPENDENT columns together to serve the purpose of Primary Key
* In MySQL, you can combine upto 32 columns in a composite Primary Key
* if you declare a composite Primary Key, then the index that is created automatically, happens to be composite unique index
* if you cannot identify some key column, then you add an extra column to the table to serve the purpose of Primary Key, such a key is known as **SURROGATE KEY**
* for SURROGATE KEY, CHAR datatype is recommended
* **YOU CAN HAVE ONLY 1 PRIMARY KEY PER TABLE**

CANDIDATE KEY -> **is not a constraint**
CANDIDATE KEY -> **is a definition**
CANDIDATE KEY **-> besides the Primary, any other column in the table that could also serve the purpose of Primary key, is a good candidate for Primary key, is known as Candidate key**

* it's good to have couple of candidate keys in your table, because in future if you Alter your table and DROP the Primary Key column, then your table is left without a Primary Key, in that situation you can make 1 of your candidate key columns as the new Primary Key

```
create table emp (empno char(4) primary key, ename varchar(25), sal float, deptno int);

create table emp (empno char(4), ename varchar(25), sal float, deptno int, primary key (deptno,empno));
```
-> composite Primary Key

```
select * from information_schema.table_constraints;

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

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

* **unique index is automatically created**

Constraints are of 2 types: -

1. Column level constraint (specified on one individuaal column)
2. Table level constraint (specified on combination of two or more columns) (composite) (has to be specified at the end of the structure)

show indexes from emp;

To drop primary key constraints: -

alter table emp drop primary key;

to add primary key constraint afterwards to an alredy existing table: -

alter table emp add primary key(deptno);

alter table emp add primary key(deptno, empno);

limitations/restrictions imposed on a table

NOT NULL

* null vaalues are not allowed (**it's a mandatory column**)
* **duplicate values are allowed**
* can have any number of not null constraints per table
* alaways a column level constraint

create table emp (empno char(4), ename varchar(25) not null, sal float not null, deptno int);

* **In MySQL, nullability is a feature of the datatype**

to see which are the not null columns: -
desc emp;

to drop the not null constraint: -
alter table emp modify ename varchar(25) null;

to add the not null constraints aafterward to an already existing table: -
alter table emp modify ename varchar(25) not null;

Solution for Candidate Key columns: -
not null constraint + unique index

ALTERNATE KEY -> for a candidate key column, if you apply a not null constraint and you create an unique index, then it works similar to Primary Key, it becomes an ALTERNATE to Primary Key, such a candidate key column is known as ALTERNATE KEY

SUPER KEY -> if you have a Primary Key and Alternate key in the table, then the Primary Key is also known as SUPER KEY

UNIQUE

* will not allow duplicate values (similar to Primary Key)
* **will allow null vaalues** (unlike Primary Key)
  (can have any number of null values)
* **TEXT and BLOB cannot be UNIQUE**
* **UNIQUE INDEX is created automatically**
* can combine **upto 32 column**s in a composite unique
* CAN HAVE ANY NUMBER OF UNIQUE KEY CONSTRAINTS

```
create table emp (empno char(4), ename varchar(25), sal float, deptno int, mob_no char(15) unique, unique (deptno,empno));

select * from information_schema.table_contraints;

select * from information_schema.table_contraints
where table_schema = 'cdac' ;

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

* unique index automatically created

show indexes from emp;
O/P :
mob_no
deptno

unique constraint is also an index, so to drop it use: -

```
drop index mob_no on emp;
drop index deptno on emp;
```

to add unique contraints afterward to an existing table: -

```
alter table emp add constraint u_emp_mob_no unique (mob_no);
```

constraint u_emp_mob_no -> constraint constraintname
constraint u_emp_mob_no -> optional

* column level constraint can be specified at table level, but a table level composite constraint can never be specified at column level
* column level constraint can be specified at table level, except for the not null constraint which is always a columnlevel and therefore the syntax will not support specifying it at the end of the structure

Day10

CONTRAINTS: -

FOREIGN KEY: -

* column or set of columns that references a column or set of columns of some table
* Foreign key constraint is specified on the child column (not the parent column)
* parent column has to be **PRIMARY Key or UNIQUE**
* Foreign Key will allow duplicate values (unless specified otherwise)
* Foreign Key will allow null values (unless specified otherwise)
* Foreign Key may reference a column of the same table also (known as self-referencing)

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	location
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

```
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

* **you can delete the parent row provided child row don't exist**

```
delete from dept where deptno =3;
```

* **you cannot delete the parent row when child rows exist**

```
delete from dept where deptno = 2;
```

ON DELETE CASCADE -> if you delete the parent row then it will automatically delete the child rows also

```
delete from dept where deptno = 2;
```

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

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

* you cannot update the parent column when child rows exist

```
update dept set deptno = 4 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 on update cascade,
constraint fk_emp_mgr foreign key (mgr) references emp(empno));
```

ON UPDATE CASCADE -> if you update the parent column then it will automatically update the child rows also

```
select * from information_schema.table_constraints;

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

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

to drop the foreign key constraint: -

```
alter table emp drop foreign key fk_emp_deptno;
```

COMPOSITE FOREIGN KEY: -

ORDER_MST

branch_cd	onum	cnum	odate	->	**Parent**
B1	1				
B1	2				
B1	3				
B2	1				
B2	2				

ORDER_DTLS

branch_cd	onum	prod_cd	qty	->	**Child**
B1	1	DVD	10000		
B1	1	USB	20000		

branch_cd & onum -> **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),
constraint constraint_fk_abc foreign key (branch_cd, onum),
constraint constraint_fk_order
```

to add foreign key constraint afterwards to an existing tables: -

```
alter table emp
add foreign key (deptno)
references dept (deptno);
```

CHECK CONSTRAINT: -

* used for validations (used for checking purposes)
e.g sal < 10000, age > 21, etc.
* Operators we can use
- Relational operator
- Arithematic operator
- Logical operator
- Special operator
  e.g between, like, in, etc.
- Call single row function

```
create table emp(
empno int,
ename varchar(25),
sal float check (sal > 5000 and sal < 150000),
deptno int,
status char(1) check (status in('T','P','R')),
comm float,
mob_no char(15),
check (sal+comm < 200000));
```

```
create table emp(
empno int auto_increment primary key,
ename varchar (25) check (ename = upper(ename)),
sal float default 7000
check (sal between 5001 and sal  149999),
deptno int,
status char(1) default 'T'
check (status in('T','P','R')),
comm float,
mob_no char(15) unique,
check (sal+comm < 200000),
constraint fk_emp_deptno foreign key (deptno) references dept (deptno));
```

DEFAULT is not a constraint
DEFAULT is a clause that you can use with CREATE TABLE

* If you specify some value, tthen it will take that value
* If **nothing is specified**, then it will take default value

to make use of DEFAULT value ant AUTO_INCREMENT, use the following INSERT statement: -

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

MySQL

* 63 system tables in MySQL
* System tables are stored in information_schema
* all system tables are read only
* e.g. statistics (for indexes) , table_constraints, key_column_usage, table_privileges, etc.

Data is of 2 types: -

1. User Data

* user created
* user tables and indexes

2. System data (also known as Metadata) (Data about data)

* MySQL created
* data that is stored in system tables

STORED OBJECTS: -

* objects that are stored in the database
* e.g. tables, indexes

VIEWS: -

* handle to a table
* stores the address of table (HD pointer) (also known as LOCATOR)
* used for indirect access to the table
* USED FOR SECURITY PURPOSES
* used to restrict the access of the users
* VIEWS are in all RDBMS and some DBMS also

***created by user edac1 & schema is cdac

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

MYSQL> create view viewname .....;

edac_mysql> create view v1 as select empno , ename from emp;

* viewname and tablename cannot be the same

edac_mysql> grant select on v1 to king;

king_mysql> select * from cdac.emp; -> **ERROR**

king_mysql> select * from cdac.v1;

OUTPUT: -

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

v1 = select empno,ename from emp;

* used to restrict the column access
* form of encapsulation (data hiding)
* **VIEW DOES NOT CONTAIN DATA**
* only the definition is stored, data is not stored
* view is a stored query
* stored in the database
* SELECT statement on which the view is based, it is stored in the DB in the COMPILED FORMAT
* view is an executable format of SELECT statement

* hence the execution will be very fast
* hiding source code from other users
* DML operations can be done on a view
* DML operations performed on a view will affect the base table
* hence the entire application is created using views
* view with check option (like having different check contraints for different users, which otherwise is not possible)
* view based on view is allowed

edac_mysql> grant select, insert on v1 to king;

king_mysql> select * from cdac.v1;

king_mysql> insert into cdac.v1 values (6, 'F');

* DML operations can be done on a view
* DML operations performed on a view will affect the base table
* hence the entire application is created using views
* constraints specified on the base table will be enforced when you INSERT via the view

To DROP the VIEW: -

edac_mysql> drop view v1;

---

edac_mysql> create view v2 as select * from emp where deptno = 1;

edac_mysql> grant select, insert on v2 to scott;

scott_mysql> select * from cdac.emp; -> **ERROR**

scott_mysql> select * from cdac.v2;

OUTPUT: -

empno	ename	sal	deptno
1	A	5000	1
2	B	6000	1
3	C	7000	1

* used to restrict the row access

---

edac_mysql> create view v2 as select * from emp
where deptno = 1 WITH CHECK OPTION;

scott_mysql> insert into cdac.v2 values (6, 'F', 6000, 2); -> **ERROR**

* view with check option (like having different check contraints for different users, which otherwise is not possible)

to change the SELECT statement on which the view is based: -

```
drop view v1;
create view v1 as select .....................;
```

```
create or replace view v1 as select ename, sal from emp;

desc emp;

desc v1;
```

```
create or replace view v1 as
select ename, sal *12 as annual from emp;

select * from v1;

insert into v1 .............................;
```

-> **ERROR**

* view based on computed column, it is recommended you specify an alias for the virtual column
* can only SELECT from this view
* DML operations are not allowed
* common for all RDBMS

Create or replace view v1 as

```
select upper(ename) as u_name, sal from emp;

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

* view based on GROUP BY clause, it is recommended you specify an alias for the virtual column
* can only SELECT from this view
* DML operations are not allowed
* common for all RDBMS

Create or replace view v1 as

```
select danem, ename from emp, dept
where dept.deptno = emp.deptno;
```

drop view v1;

* **if you drop the table, the views remains but as invalid**

show tables; -> will show tables and views but it won't tell which is a table and which is a view

To see which is a table and which is a view: -

show full tables;

To see the SELECT statement of v1: -

show create view v1;

* view based on view is allowed

USES: -

* to exceed 255 levels limit of function within function
* to exceed 255 SELECT statements limit for SET operators

e.g. UNION of 300 SELECTs

* to exceed 255 levels limit of sub-queries
* to simplify the writing of complex SELECT statements

e.g. JOIN of 20 tables

* complex queries can be stored in view definition

(e.g. join of 10 tables)

* to convert 3D table into 2D table
* to convert 2D table into 3D table
* to apply Relational methods on Object tables
* to apply Object methods on Relational tables
* Data mapping
* Data migration
* EAI (Enterprise Application Integration)
* EIM (Enterprise Integration Management)

MySQL PL Programming Language

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 in MySQL command line client, MySQL Workbench, Oracle Forms, Oracle Reports, Oracle Menus, Oracle Graphics, Oracle Apex, Java, etc.

* Few 4 GL features (supports few OOPS features)

```
Begin                    ->      start of program

insert into dept values(a, 'a', 'B');
                         //MySQL PL  Block

End;                     ->      end of program
```

Block within a Block: - (**Execution is TOP to BOTTOM**)

* **Block level language** (feature of OOPS)

```
Begin
.................
Begin
.....................
.....................
End;
.................
End;
```

Benefits of Block level language: -

a. Modularity

b. Control scope of variables (form of encapsulation/data hiding)

c. Exceptions to localize the error (efficient error management)

* screen input and screen output is not allowed (scanf, printf, etc are not available)

* used ONLY for processing

* can use SELECT statement inside the block but it's **not recommended**

* **SQL commands that are allowed in MySQL PL are : -**
  DDL, DML, DQL, DTL/TCL

* **DCL commands not allowed inside the MySQL PL block**

to store output of MySQL PL program: -

```
create table tempp(
fir int,
sec char(15));
```

TEMPP

FIR	SEC

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

STORED OBJECTS: -

* objects that are stored in the databse

* e.g. tables, indexes, views

STORED PROCEDURES

* Routine (set of commands) that has to be called explicitly
* global procedures
* can be called from MySQL Command Line Client, MySQL Workbench, etc.
* can be called through **any front-end s/w**
* stored in the databse in the COMPILED FORMAT
* hence the execution will be very fast
* hiding source code from end user
* stored procedure can have local variables, cursors, etc.
* within procedure all MySQL commands
  e.g. IF statements, loops, etc.
* one procedure can call another procedure
* procedure can call itself **(known as RECURSION**)
* procedure can have parameters
* OVERLOADING OF PROCEDURE IS NOT ALLOWED (you cannot have 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)
* In a multi-user environment, if multiple users are calling the same procedure simultaneously, then only a single copy of the procedure code is brought into the server RAM (procedure code will be shared by the users)

```
create table tempp(
fir int,
sec char(15));
```

PROGRAM 1: -

```
delimeter //
```
-> entire procedure will be treated as one unit (delimeter can be any special char such as (//,?, ***, etc)

```
create procedure abc()
begin
insert into tempp values(1, 'inside abc');
end; //

delimeter ;
```
-> **change to original**

-->> Read, Compile ,Plan, Store it in the DB in the COMPILED FORMAT

to call the stored procedure: -

```
call abc();
```

to see the output of procedure: -

```
select * from tempp;
```

OUTPUT:-

TEMPP

FIR	SEC
1	inside abc

if you don't want procedure in future then you can drop it: -

```
drop procedure abc;
```

Day11

```
create table tempp(
fir int,
sec char(15));
```

TEMPP

FIR	SEC

PROGRAM 2: -

```
delimiter //
create procedure abc()
begin
declare x int;                  ->          scope of x is limited to this block (local variable)
set x = 10;
insert into tempp values(x, 'inside abc');
end; //
delimiter ;
```

* In MySQL PL, when you declare a variable, if you don't initialize it,it will store a null value

* You can declare a variable and assign a value simultaneously

```
delimiter //
create procedure abc()
begin
declare x int default 10;
insert into tempp values(x, 'inside abc');
commit;                                         ->          optional
end; //
delimiter ;
```

PROGRAM 2: -

```
delimiter //
create procedure abc()
begin
declare x char(15) default 'CDAC';
insert into tempp values(1, x);
end; //
delimiter ;
```

OUTPUT: -

TEMPP

FIR	SEC
1	CDAC

PROGRAM 3: -

Write a program for HRA calculation:-

*HRA = 40% of sal

```
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 ;
```

OUTPUT: -

TEMPP

FIR	SEC
3000	KING
1200	HRA

```
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 ;
```

-> PARAMETERIZED Procedure

* You can pass parameters to a procedure

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

-- **Single Line Comment**
//** **Multiline Comment**

EMP

ename	sal	job
SCOTT	3000	CLERK
KING	5000	MANAGER

```
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 */
insert into tempp values(x , 'KING');
end; //
delimiter ;
```

```
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 */
insert into tempp values(x , 'KING');
end; //
delimiter ;
```

call abc('KING');
call abc('SCOTT');

```
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 ;
```

drop procedure abc;

to see which all procedures are available: -

show procedure status; -> shows all procedures in all schemas

show procedure status where db = 'cdac' ;

show procedure status where name like 'A%';

to view the source code of store procedure: -

show stored procedure abc;

to share the procedure with other users: -

edac_mysql> grant execute on procedure abc to scott;

scott_mysql> call cdac.abc();

edac_mysql> revoke execute on procedure abc from scott;

Decision making using IF statement: -

EMP

ename	sal
KING	5000

```
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 ;
```

```
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 ;


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 ;
```

ELSEIF construct: -

```
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 ..... then
..... ;
elseif..... then
.....;
elseif..... then
.....;
elseif..... then
.....;
elseif..... then
.....;
end if;
```

```
if x > 5000 and x < 6000 then                    (and, or)
..... ;
elseif y like 'A%' then                    (like, in, between)
.....;
elseif..... then
.....;
elseif..... then
.....;
elseif..... then
.....;
end if;
```

```
delimiter //
create procedure abc()
begin
declare x boolean default TRUE;
if x  then
        insert into tempp values(1 , 'Mumbai');
end if;
end; //
delimiter ;
```

OUTPUT: -

TEMPP

FIR	SEC
1	Mumbai

```
delimiter //
create procedure abc()
begin
declare x boolean default FALSE;
if not x then
        insert into tempp values(1 , 'Delhi');
end if;
end; //
delimiter ;
```

OUTPUT: -

TEMPP

FIR	SEC
1	Delhi

LOOPS: - (for repetitive/iterative processing)

WHILE loop: -

* check for condition before entering the loop

Syntax: -

```
WHILE expression DO
.....;
.....;
END WHILE;
```

```
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 ;
```

OUTPUT: -

TEMPP

FIR	SEC
1	in while loop
2	in while loop
3	in while loop
4	in while loop
5	in while loop
6	in while loop
7	in while loop
8	in while loop
9	in while loop

NESTED WHILE loop: -

```
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 ;
```

OUTPUT: -

TEMPP

FIR	SEC
1	in y loop
2	in y loop
3	in y loop
4	in y loop
5	in y loop
6	in y loop
7	in y loop
8	in y loop
9	in y loop
1	in x loop
2	in x loop
3	in x loop
4	in x loop
5	in x loop
6	in x loop
7	in x loop
8	in x loop
9	in x loop

```
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 ;
```

OUTPUT: -

TEMPP

FIR	SEC
1	in x loop
1	in y loop
2	in x loop
2	in y loop
3	in x loop
3	in y loop
4	in x loop
4	in y loop
5	in x loop
5	in y loop
6	in x loop
6	in y loop
7	in x loop
7	in y loop
8	in x loop
8	in y loop
9	in x loop

REPEAT loop: - (similar to DO-WHILE loop)

* it will execute at least once

Syntax: -

```
REPEAT
.....;
.....;
UNTIL expression_is_not_satisfied
END REPEAT;
```

```
delimiter //
create procedure abc()
begin
declare x int default 1;                    (try for x = 100)
repeat
        insert into tempp values(x , 'in loop');
        set x = x+1;
until x > 5
end repeat;
end; //
delimiter ;
```

OUTPUT: -

TEMPP

FIR	SEC
1	in loop
2	in loop
3	in loop
4	in loop
5	in loop

Loop, Leave and Iterative statements: -

* **Leave** statement allows you to exit the loop (similar to 'break' statement)

* Iterate statement allows you to skip the entire code under it, and start a new iteration (similar to 'continue' statement)

* Loop statement executes a block of code repeatedly with an additional flexibilty of using LOOP LABEL (you can give a name to a loop)

Program :-

```
delimiter //
create procedure abc()
begin
declare x int default 1;
pqr_loop:loop                          ->      LABEL
        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 , 'inside loop');
        end if;
end loop;
end; //
delimiter ;
```

OUTPUT: -

TEMPP

FIR	SEC
2	inside loop
4	inside loop
6	inside loop
8	inside loop
10	inside loop

Session Variables: -

* Global variables

* create and initialize simultaneously

* available in the server RAM till you end your session

* you can manupulate session variables

```
mysql> set @x = 10;

mysql> select @x from dual;                    ->        10


insert into tempp values (@x, 'Hello World!');

select * from tempp;

Mysql > @x = x+1;
```

* **Works in MySQL Command Line and Workbench also**

Day12

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

TEMPP

fir	sec

CURSORS: - (Most IMP)

* present in all RDBMS, some DBMS, and some front-end s/w's also
* CURSOR is a type of a variable
* CURSOR can store multiple rows
* CURSOR is similar to 2D ARRAY
* CURSORS are used for processing multiple rows
* CURSORS are used for storing multiple rows
* CURSORS are used for handling multiple rows
* CURSORS are used for storing the data temporarily
* CURSOR is based on SELECT statement in MySQL
* CURSOR is a READ_ONLY variable
* you will have to fetch 1 row at a time into some intermediate varaibles and do your processing with those variables
* can only fetch sequentially (top to bottom)
* YOU CANNOT FETCH BACKWARDS IN A CURSOR
* can only fetch 1 row at a time

```
delimiter //
create procedure abc()
begin
declare a int;
declare b varchar(15);
declare c int;
declare d int;
declare x int default 1;
declare c1 cursor for select * from emp;   ->      CURSOR Declaration/Definition
open c1;                                           ->      opens the CURSOR and fires the SELECT statement
while x < 6 do                                (try x < 4, x < 11)
        fetch c1 into a,b,c,d;
        /* processing, e.g. set hra_calc = c*0.4, etc
        update emp set hra = hra_calc where empno = a */
        insert into tempp values(a, b);
        set x = x + 1;
end while;
close c1;                                          ->      will close the cursor and it will free the RAM
end; //
delimiter;
```

CURSOR C1

empno	ename	sal	deptno
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

OUTPUT: -

TEMPP

fir	sec
1	A
2	B
3	C
4	D
5	E

```
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;
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: -

```
delimiter //
create procedure abc()
begin
declare a int;
declare b varchar(15);
```

```
declare c int;
declare d int;
declare finished int default 0;
declare c1 cursor for select * from emp;
declare continue handler for not found set finished = 1;
open c1;
cursor_c1_loop : loop
        fetch c1 into a,b,c,d;
        if finished = 1 then
                leave cursor_c1_loop;
        end if;
        insert into tempp values(a, b);
end loop cursor_c1_loop;
close c1;
end; //
delimiter;
```

* **NOT FOUND IS A CURSOR ATTRIBUTE, IT RETURNS A BOOLEAN TRUE VALUE IF THE LAST FETCH WAS UNSUCCESSFUL**

```
delimiter //
create procedure abc()
begin
declare a varchar(15);
declare b int;
declare finished int default 0;
declare c1 cursor for select ename, sal from emp;
declare continue handler for not found set finished = 1;
open c1;
cursor_c1_loop : loop
        fetch c1 into a,b;
        if finished = 1 then
                leave cursor_c1_loop;
        end if;
        insert into tempp values(b, a);
end loop cursor_c1_loop;
close c1;
end; //
delimiter;
```

CURSOR C1

ename	sal
A	5000
B	6000
C	7000
D	9000
E	8000

OUTPUT: -

TEMPP

fir	sec
5000	A
6000	B
7000	C
9000	D
8000	E

* you cannot open the same cursor repeatedly
* you will have to close the cursor before you can open it again

to reset the cursor pointer: -

```
close c1;
open c1;
```

```
delimiter //
create procedure abc()
begin
declare a int;
declare b varchar(15);
declare c int;
declare d int;
declare finished int default 0;
declare c1 cursor for select * from emp where deptno = 1;
declare continue handler for not found set finished = 1;
open c1;
cursor_c1_loop : loop
        fetch c1 into a,b,c,d;
        if finished = 1 then
                leave cursor_c1_loop;
        end if;
        insert into tempp values(a, b);
end loop cursor_c1_loop;
close c1;
end; //
delimiter;
```

```
delimiter //
create procedure abc()
begin
declare a varchar(15);
declare b int;
declare finished int default 0;
declare c1 cursor for select lower(ename) as l_ename, sal+500 as bonus from emp;
declare continue handler for not found set finished = 1;
open c1;
```

```
cursor_c1_loop : loop
        fetch c1 into a,b;
        if finished = 1 then
                leave cursor_c1_loop;
        end if;
        insert into tempp values(b, a);
end loop cursor_c1_loop;
close c1;
end; //
delimiter;
```

CURSOR C1

l_name	bonus
a	5500
b	6500
c	7500
d	9500
e	8500

OUTPUT: -

TEMPP

fir	sec
5500	a
6500	b
7500	c
9500	d
8500	e

DEPT

deptno	dname	location
1	TRN	Bby
2	EXP	Dlh
3	MKTG	Cal

```
delimiter //
create procedure abc()
begin
declare a varchar(15);
declare b int;
etc.
declare finished int default 0;
declare c1 cursor for select * from dept;
declare c2 cursor for select * from dept;
declare continue handler for not found set finished = 1;
open c1;
```

```
open c2;
cursor_c1_loop : loop
        fetch c1 into a,b;
        if finished = 1 then
                leave cursor_c1_loop;
        end if;
        insert into tempp values(a, b);
end loop cursor_c1_loop;
close c1;
end; //
delimiter;
```

* **IN MySQL, NO UPPER LIMIT ON THE NUMBER OF CURSORS THAT CAN BE OPENED AT A TIME**

```
delimiter //
create procedure abc()
begin
declare a varchar(15);
declare b int;
etc.
declare finished int default 0;
declare c1 cursor for select empno, dname from emp, dept
where dept.deptno = emp.deptno;
declare continue handler for not found set finished = 1;
open c1;
cursor_c1_loop : loop
        fetch c1 into a,b;
        if finished = 1 then
                leave cursor_c1_loop;
        end if;
        insert into tempp values(a, b);
end loop cursor_c1_loop;
close c1;
end; //
delimiter;
```

CURSOR C1

empno	dname
1	TRN
2	TRN
3	TRN
4	EXP
5	EXP

OUTPUT: -

TEMPP

fir	sec
1	TRN
2	TRN
3	TRN
4	EXP
5	EXP

```
delimiter //
create procedure abc()
begin
declare a int;
declare b varchar(15);
declare c int;
declare d int;
declare finished int default 0;
declare c1 cursor for select * from emp;
declare continue handler for not found set finished = 1;
open c1;
cursor_c1_loop : loop
        fetch c1 into a,b,c,d;
        if finished = 1 then
                leave cursor_c1_loop;
        end if;
        update emp set sal = sal + 1;
end loop cursor_c1_loop;
close c1;
end; //
delimiter;
```

CURSOR C1

empno	ename	sal	deptno
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

* **ABOVE PROGRAM WILL UPDATE THE SAL COLUMN BY +5**

```
delimiter //
create procedure abc()
begin

declare a int;
declare b varchar(15);
declare c int;
declare d int;

declare finished int default 0;

declare c1 cursor for select * from emp;

declare continue handler for not found set finished = 1;

open c1;

cursor_c1_loop : loop
        fetch c1 into a,b,c,d;
        if finished = 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;
```

* **ABOVE PROGRAM WILL UPDATE THE SAL COLUMN BY +2**

```
delimiter //
create procedure abc()
begin
declare a int;
declare b varchar(15);
declare c int;
declare d int;
declare finished int default 0;
declare c1 cursor for select * from emp for update;          ->        LOCKS THE ROWS
declare continue handler for not found set finished = 1;
open c1;
cursor_c1_loop : loop
        fetch c1 into a,b,c,d;
        if finished = 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;                                ->        LOCKS ARE AUTOMATICALLY RELEASED WHEN YOU ROLLBACK OR
                                                 COMMIT
end; //
delimiter;
```

* **ABOVE PROGRAM WILL UPDATE THE LAST 2 ROWS OF SAL COLUMN BY +1**

```
delimiter //
create procedure abc()
begin
declare a int;
declare b varchar(15);
declare c int;
declare d int;
declare finished int default 0;
declare c1 cursor for select * from emp for update;
declare continue handler for not found set finished = 1;
open c1;
cursor_c1_loop : loop
        fetch c1 into a,b,c,d;
        if finished = 1 then
                leave cursor_c1_loop;
        end if;
        if c > 7000 then
                delete from emp where empno = a;
        end if;
```

```
end loop cursor_c1_loop;
close c1;
commit;
end; //
delimiter;
```

* **ABOVE PROGRAM WILL DELETE THE LAST 2 ROWS**

Types of CURSORS: -

1. EXPLICIT CURSOR

* user/programmer created
* have to be declared explicitly
* used for storing/processing multiple rows
* **USED TO LOCK THE ROWS MANUALLY**

BEFORE YOU ISSUE UPDATE OR DELETE, YOU SHOULD LOCK THE ROWS MANUALLY: -

* TO LOCK THE ROWS MANUALLY, **YOU WILL REQUIRE A CURSOR WHOSE SELECT STATEMNT IS HAVING A FOR UPDATE CLAUSE;** SIMPLY OPEN THE CURSOR AND THEN CLOSE IT; THE ROWS OF THE TABLE WILL REMAIN LOCKED TILL YOU ISSUE A ROLLBACK OR COMMIT: -

```
.....;
.....;
declare c1 cursor for select * from emp for update;
open c1;
close;
.....;
```

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

2. IMPLICIT CURSOR

* not available in MySQL
* available in Oracle
* Oracle created

Procedures Parameters are of 3 types: -

IN **(BY DEFAULT)**

* Read only
* can pass constant, variable, expression
* call by value
* **FASTEST i**n terms of processing speed

PROGRAM :

```
delimiter //
create procedure abc(in y int)                    ->    in is optional
begin
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);
```

OUTPUT: -

TEMPP

fir	sec
5	inside abc
10	inside abc
25	inside abc

OUT (SLOW compared to IN) (MOST SECURE)

* Write only
* can pass variables only (constants and expressions are NOT ALLOWED)
* call by reference
* procedure can return a value indirectly if you call by reference
* used on public network

```
delimiter //
create procedure abc(out y int)
begin
set y = 100;
end; //
delimiter ;

set @x = 10;
select @x from dual;          ->    10

call abc(@x);                 ->    address is passed not value
select @x from dual;          ->    100
```

INOUT (SLOW compared to IN) (MOST POWERFUL)

* Read and Write
* can pass variables only (constants and expressions are NOT ALLOWED)
* call by reference
* procedure can return a value indirectly if you call by reference
* used on local network

```
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);                 ->    address is passed not value
select @x from dual;          ->    1000
```

STORED OBJECTS: -

* objects that are stored in the database

e.g. create..... tables, indexes, views, procedures, functions

STORED FUNCTIONS: - (STORED OBJECTS)

* Routine that returns a value directly and compulsorily
* global functions
* can be called from any front-end s/w
* stored in the databse in the COMPILED FORMAT
* hence the execution will be very fast
* hiding source code from end user
* etc. benefits same as procedures
* IN PARAMETRES ONLY

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

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

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

call pqr();

OUTPUT: -

TEMPP

fir	sec
10	after abc

```
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();

OUTPUT: -

TEMPP

fir	sec
100	after abc

INTERVIEW QUESTION: -

whats is similarity between stored procedure and stored function?

whats is difference between stored procedure and stored function?

- stored function can be called in select statement
- stored function can be called in SQL statements

select abc(sal) from emp;
select abc(10) from dual;
delete from emp where abc(sal) = 100000;

```
delimiter //
create function abc(y int)
returns int
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 ;
```

EMP

ename	sal
KING	9000

call pqr();

OUTPUT: -

TEMPP

fir	sec
9000	> 5000

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;

Day13

DATABASE TRIGGERS (V. Imp) (**Stored Objects**)

* 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: -**
  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

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

OUTPUT: -

deptno	sum(sal)
1	15000
2	6000

```
select * from deptot;
```

OUTPUT: -

deptno	saltot
1	15000
2	6000

```
delimiter //
create trigger abc
before 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

* MySQL will read, compile, make aplan, and store it in the databse 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

* ROLLBACK and COMMIT not allowed inside the trigger

* **ROLLBACK or COMMIT is to be specified AFTERWARDS, at the and of transaction**

* whether you COMMIT or ROLLBACK afterwards, the data will always be consistent

* 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

* In MySQL all triggers are at ROW LEVEL (they will fire for each row)

* **In MySQL you can have max 6 triggers per table**

```
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 databse in the case of Insert

* maintain SHADOW tables in the event of insert

```
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 ;
```

USES: -

* automatic updation of related tables

to drop the trigger: -

```
drop trigger abc;
```

******if you drop the table, then indexes and triggers are dropped automatically (view,procedures remains)**

```
show triggers;   ->        shows all triggers in all schemas
show triggers from [db_name];
show triggers from cdac;
select * from information_schema.triggers;
```

---

delete from emp where deptno = 2;

```
delimiter //
create trigger abc
before delete
on emp for each row
begin
insert into tempp values(1, 'deleted', user(), sysdate());
end; //
delimiter ;
```

USES: -

* maintain logs (AUDIT TRAILS) of deletions

```
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

```
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 ;
```

---

update emp set sal = 6000 where deptno = 1;

```
delimiter //
create trigger xyz
before update
on emp for each row
begin
insert into tempp values(1, 'updated');
end; //
delimiter ;
```

USES: -

* maintain logs (AUDIT TRAILS) of updations

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**

MUTATING TABLES ERROR -> if some casscading trigger tries tries to perform any DML operation on one of the previous tables, you will get an error of Mutating tables and the entire transaction is automaticlly ROLLED BACK

```
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 xyz
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

update emp set sal = 6000 where ename = 'A';

```
delimiter //
create trigger xyz
before update
on emp for each row
begin
update deptot set saltot = saltot - old.sal + new.sal
where deptno = old.deptno;
end; //
delimiter ;
```

NORMALISATION: - (V. IMP)

* only available in RDBMS
* concept of table design
* **Primary Key is a by-product of Normalisation**
* what table to create, structures, columns, datatypes,widths, constraints
* based on user requirements
* part of design phase
* aim is to have an "efficient" table structure, to avoid Data Redundancy (avoid the unnecessary duplication of data)
* aim is to reduce the problems of insert, update and delete

Traditional approach: -

* aim was to allow the simple retrieval of data
* Normalisation was done from an outer perpestrive
* Normalisation was done from a reports perpestrive

Modern approach: -

* aim is to reduce the problems of insert, update, and delete
* Normalisation done from an input perspective
* Normalisation done from a forms perspective
* VIEW THE ENTIRE APPLICATION ON A PER-TRANSACTION BASIS AND YOU NORMALISE ECH TRANSACTION SEPERATELY

e.g.

CUSTOMER_PLACES_AN_ORDER,CUSTOMER_MODIFIES_AN_ORDER,
CUSTOMER_MAKES_PAYMENT,
GOODS_ARE_DELIVERED, etc.

Getting ready for NORMALISATION: -

1. Select a Transaction (e.g. CUSTOMER_PLACES_AN_ORDER)
2. Ask the Client for sample data
3. For a transaction, make a list of all the fields
4. For all practical purposes, we can have a single table with all these columns
5. Taking client into confidence, Strive for atomicity (column can be broken up into sub-columns)
6. For every column, make a list of column properties
7. Get User sign-off
8. End of User involvement
9. For all practical purposes, we can have a single table with all these columns
10. Assign Datatypes and widths
11. Specify not null, unique and check constraints
12. Remove the computed columns
13. **Key element will be Primary Key of this table**

* **At this point, data is in Un-Normalised Form (known as UNF)**

Un-Normalised Form -> Starting point of Normalisation

Steps of NORMALISATION: -

1. Remove the repeating group into new table
2. **Key element will be Primary Key of new table**
3. Add the Primary Key of original table to new table to give you a composite Primary Key -> this step may or maynot be required

Above the 3 steps are to be repeated infinitely till you cannot Normalise any further

FIRST NORMAL FORM: - (FNF) (1NF) (Single Normal Form)

Repeating groups are removed from table design
One : Many relationship is always encountered here
25%

4. Only the tables with composite Primary Key are examine
5. Those non-key elements that are not dependent on entire composite Primary key, they are to be removed into 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

Above the 3 steps are to be repeated infinitely till you cannot Normalise any further

SECOND NORMAL FORM: - (SNF) (2NF) (Double Normal Form)
Every column is functionally dependent on primary key (it's known as Functional Dependency)
Functional Dependency -> without Primary Key that column cannot function
67%

7. Only the non-key elements are examined
8. Inter-dependent columns are to be removed into a new table
9. Key element will be Primary Key of new table, and the Primary Key column(s) of new table is/are to be retained in the original table for relationship purposes
Above the 3 steps are to be repeated infinitely till you cannot Normalise any further

THIRD NORMAL FORM: - (TNF) (3NF) (Triple Normal Form)
Transistive dependencies (inter-dependencies) are removed from table design

FORTH NORMAL FORM: - (may or may not implement)

* also known as BCNF **(Boyce-Codd Normal Form)**
* extension of THIRD NORMAL FORM
* you may or may not implement
* used to protect the integrity of data
* normally used on public network (e.g. Internet)

Oracle
RDBMS + OODBMS -> ORDBMS

DE-NORMALISATION

* if the dta 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