# Complete DBMS and SQL

## Evolution of Databases

960 → Hierarchical model by IBM with was called Information management System by NASA
→ Network model by charles Bachman

970 → Relational model by Ted Codd

980 → DBMS launched by Oracle

1990-2000 → Advent of SQL

### What Can DBMS Do

Store, fetch, modify and delete data

Enable Concurrent access
↳ many employees can update their information in DB simultaniously.

use Transactions to ensore data Consistancy

Ensure data Recovery
↳ after Crashing data we can able to recover data.

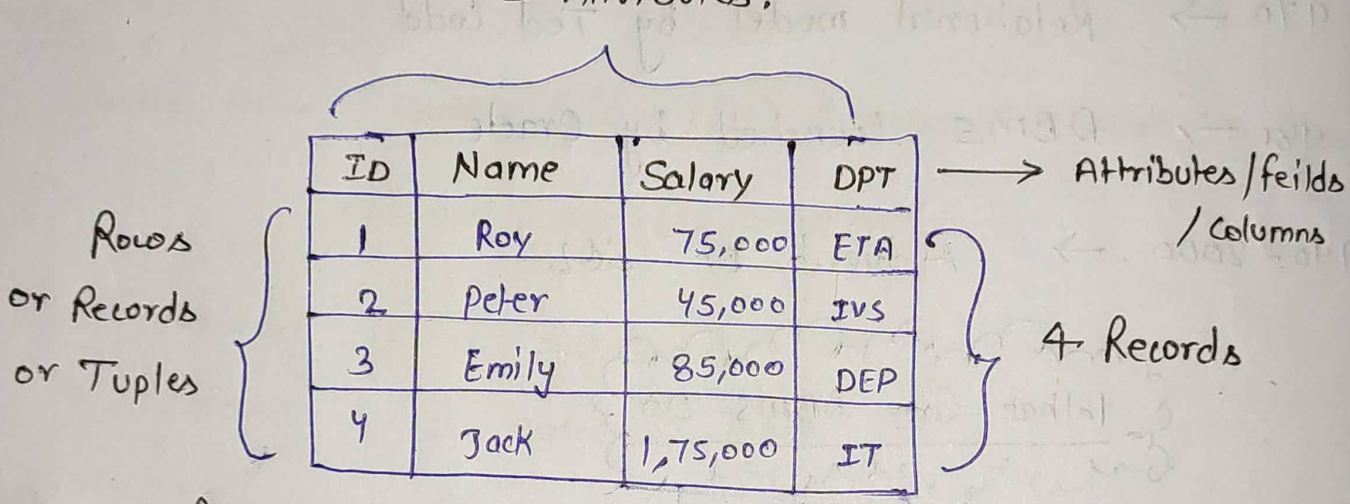
Ensure data Integrity
↳ It ensures that all data in database is valid as per the business Rules
Exa- date of Birth
Can not be a future date.

Ensure Data Security
↳ This control who can control what data

Provide utilities to administer database.
↳ we can export data from database in excel sheet

## What is RDBMS

- Data are organized in 2 Dimensinal table & and Relationship is maintained by Common feild.
- Relational model is form of table with Row and Column.
- Table are Known as Relations.
- most widely used.



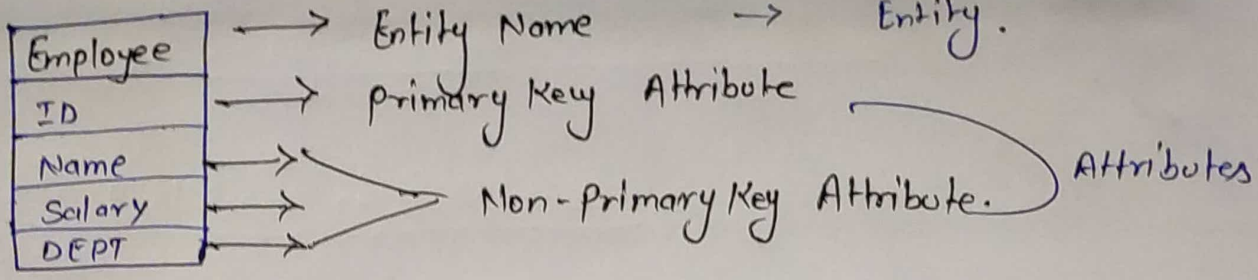

- Number of records in a relation = Cardinality of Relation.
  - ⇒ Here Cardinality of Relation = 4
- Number of attributes in relation = Degree of Relation.
  - ⇒ Here Degree of relation = 5

⇒ **Entity :-** Real word object which have and independent existence and about which we intend to Collect

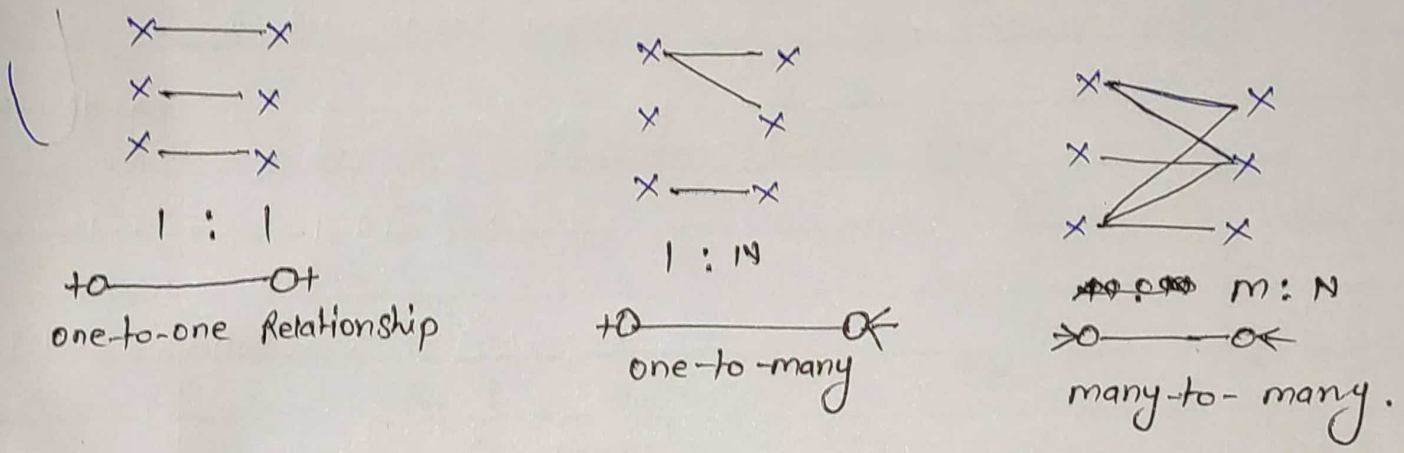
Exa Employee, Computer.

⇒ **Attribute :-** A property that describes an Entity.

Exa Name, Salary.

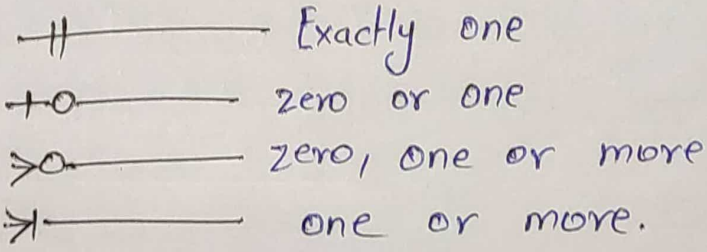


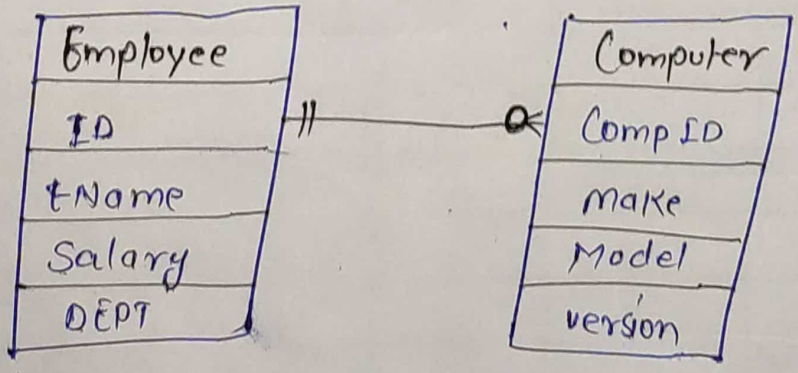

⇒ Cardinality of relationships are three types




⇒ Crow foot notation is one of the approaches to represent the Cardinality of Relationship in an ER model.

The notation comprises of four symbols and one of them needs to be used for each entity in a relationsh

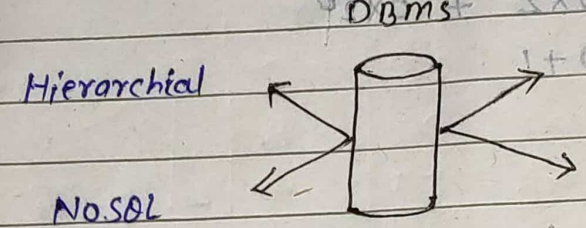


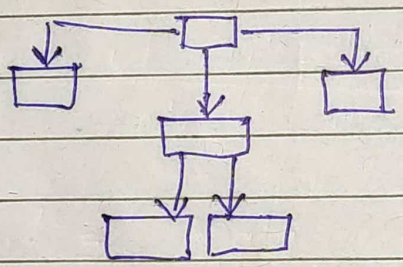
# Database management System

Date:
Page No:

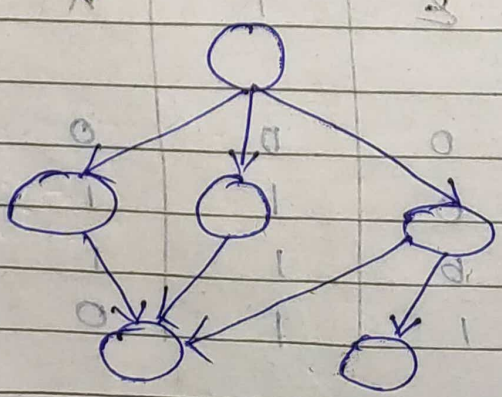
Types of DBMS - There are four types of DBMS

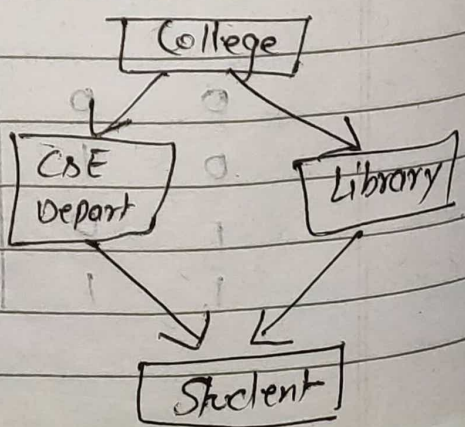



1. Hierarchical -: Hierarchical Database organize data int a tree-like structure. Data is stored as records which are connected to one another through parant chield relationships. Some examples of Hierarchical Database are information management System (Ims), Raima Database manager (RDM) mobile etc.

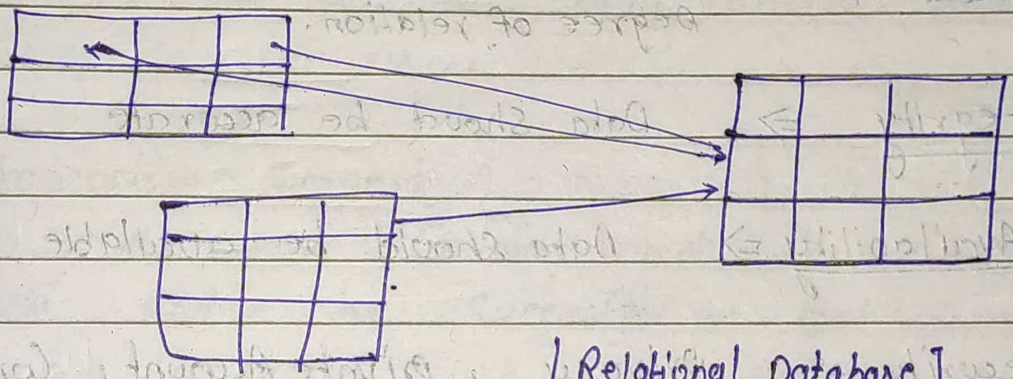


2. Network Database -: Network Database organize data into a graph structure in which object types are nodes relationship types are arcs. Each record can have multiple parent and child records. Some examples of Network Databases are integrated Database management System (IDMS), Integrated & data store (IDS) etc.



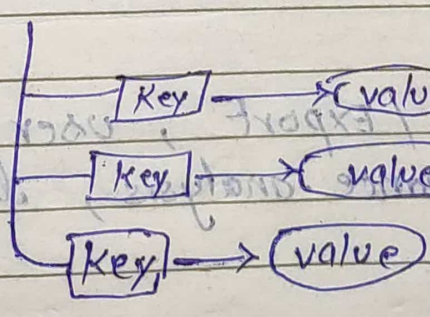



3. Relational Database :- Relational Databases organizes data into one or more tables. A table consist of attributes (columns) tuples (rows) and provides a "way" to unequaly identify each tuple. Tables are related to each other through parent child relationships. Some example of Relational Databases are DB2, oracal, SQL Server etc.



[Relational Database]

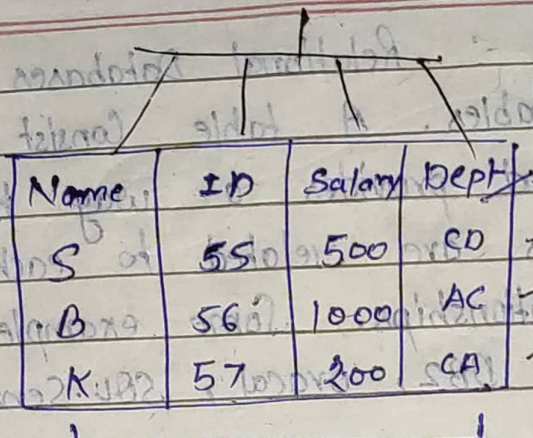
4. NoSQL Database :- NoSQL Database uses key-value, graph or document data structures to store data. These databases aim for simplicity of designing, horizontal scaling and finer control over availability. Some examples on NoSQL databases are Cassandra, mongo DB, Couch DB, orient DB, HBASE etc.




[NoSQL]

Attribute / Column / Feilds.

Ex




⇒ ✯ Integrity ⇒ Data shoud be accurate

⇒ ✯ Availability ⇒ Data should be available all time.

⇒ ✯ Security ⇒ privacy, private Account, Control in our

⇒ ✯ Independent of Application ⇒ Data can be access Android as well as mac

⇒ ✯ Concurrent ⇒ Data should be public and many people could be access my data at the same time.

① ⇒ Utilities →

Data import / Export, user management backup, performance analysis, logging and audit

② ⇒ Integrity ⇒

maintain accuracy of data.

(3) Security :- Access to Authorised users only.

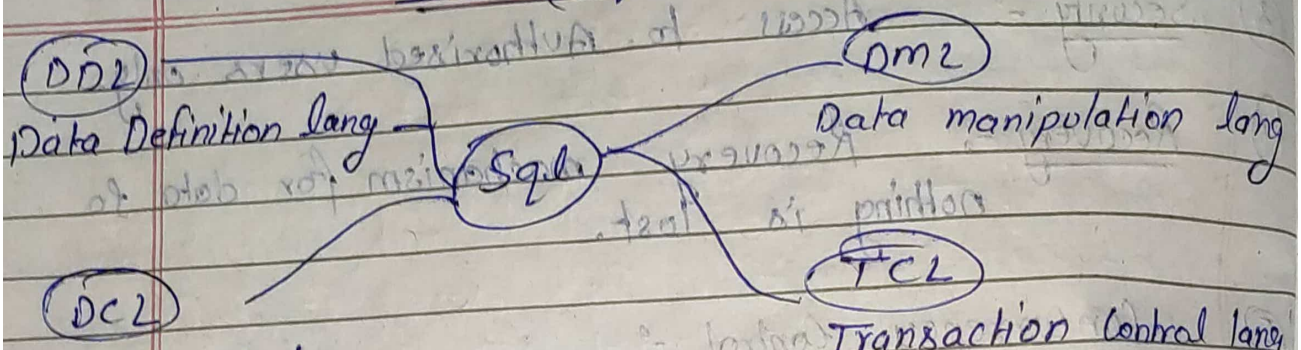
(4) Recovery :- Recovery mechanism for data so nothing is lost.

(5) Concurrency Control :- Simultaneous data access provided to users

(6) Transaction Support :- Ensure modifications to database must either be successful or not done at all.

(7) Data management :- Stores, retrieves and modify data.

# DBMS (Sql)




① **DDL**

Data Definition language is used to specify the structure, schema of a relational database.

Task performe :-

→ Create
create a new database object

→ ALTER
modify existing database object

→ DROP
Delete existing database object

→ TURNCATE
Remove all Rows from table.

② **DML**

Data manipulation language enables users to access or manipulate data in a relational database.

Task performe :-

→ INSERT :-
Create a new row in tables

→ UPDATE
modify data in tables

→ DELETE
Delete data from tables

→ SELECT
Retrive data from tables.

3. DCL
Data Control language enables user to provide access to various database like views, table, stored Procedurs etc.
Task perform.

→ GRANT
provide access rights on database.

→ ~~withdraw access~~

→ REVOKE
withdraw access rights on database.

4. TCL
Transaction Control language Specifies commands for beginning and ending a transaction.
A transaction Consist of a sequence of sql statements that are applied in an atomic (all or none) manner.

→ Commit -
- Save database changes and end transaction.

→ ROLLBACK -
Undo changes that are not Commited and end transaction.

## SQL Data Types

1. DATE — DD-MON-YY
2. TIMESTAMP
3. CLOB (character large object) - store Images limit is 400 Bites
4. BLOB (Binary large object) - Store large movie, Images upto 4GB.

| Input | Datatype | Stored value |
|---|---|---|
| 121.79 | Number | 121.79 |
| 121.75 | NUMBER(3) | 122 |
| 121.79 | NUMBER(5,2) | 121.79 |
| 121.79 | NUMBER(6,1) | 121.8 |
| 121.79 | Number (4,2) | error |

## Precision & Scale

Nonintegral data types have an integer part and a functional part Either NUMERIC, DECIMAL or Number datatype Can be used to store nonintegral number.

NUMBER(3,1)

precision Scale

Number(3) : 999.0
precision scale

Number(3,1) : 99.9

Number(3,2) : 9.99

Number(3,3) : 0.999

Scale is the number of digits allowed after the decimal

precision is the total number of Significant digits

## Operators

Airthmatic operators :

| | |
|---|---|
| Addition | + |
| Substraction | - |
| multiplication | * |
| Division | / |

---

| | |
|---|---|
| Equal to | = |
| Not Equal to | <> |
| Greater than | > |
| Greater than equal to | >= |
| less than | < |
| less than equal to | <= |

Comparision Operator.

---

Other Comparision Operator :

| operator | Symbol | Example. |
|---|---|---|
| Range | BETWEEN <lower limit> and <upper limit> | Salary Between 2500 and 3000 |
| List | IN (List of values) | Dept in('IVS', 'ETA', 'ICP') |
| String pattern matching | LIKE | SupplierId Like 'S%' |
| NULL Test | IS NULL | Bonus is null |

---

Logical operators

| operator | Example |
|---|---|
| AND | Salary >= 30000 AND Dept = 'ETA' |
| OR | Salary > 75000 OR DEPT = 'ICP' |
| NOT | ID Not IN (2,3) |

Create Table:

```
Create table Student (
    Studentid Integer Constraint Stu-PK Primary KEY,
    FName varchar2 (10),
    ContactNo NUMBER (10));
```

Drop

→ DROP Table Student;

⇒

```
Create Table Student (
    StudentID Integer CONSTRAINT Stud-PK Primary Key,
    FName VARCHAR2 (10),
    ContactNum Number (10));
// Executed the Create statement for Student (Perent)

Create Table Marks (
    CourseID Integer,
    StudentID Integer Contraint marks_FK References Student
    markScored DECIMAL (5,2));          (StudntID),
// Execute Create statement for marks (chield table)
```

⇒ Drop parant.

```
Drop Table Student CASCADE Constraints;
```

⇒ Drop chield

```
Drop Table & marks CASCADE ~~marks~~ Constraints;
```

**Note**

with out Drop - CASCADE CONSTRAINTS we Cant delete the parant table first because it's a forign key. but we can drop marks table first then perant table (Student).

# Constraints

| Constraint Type | Applies on |
|---|---|
| Single Column Constraint | Single Column |
| Composit Constraint | multiple Column |

| Constraint Type | Specified |
|---|---|
| Column level Constraint | with Column definition |
| Tevel level Constraint | After Column definition. |

Table level Constraint Can be Specified after all Column used in the Constraint have been defined. It is not neccessory t ~~Specified~~ specify them after all Column in the table are defined Composite Column Can only be Specified as table level Constrai

There are Some Constraints

- NOT NULL
- PRIMARY KEY → ID Integer Constraint Stud_PK PRIMARY KEY,
- CHECK → Gender char(1) Constraint g_CK CHECK (Gender IN ('m','f'));
- UNIQUE → ContactNo Number(10) Constraint Con_UK UNIQUE,
- FOREIGN KEY → ID Integer Constraint marks_FK References Student(studentID
- DEFAULT → Create Table student ( Id Integer, fname varchar2(

DOJ DATE Default SysDATE)

(or)

ID Integer PRIMARY Key;

(FName, Lname) CHECK (fname <> lname);

personID Integer References person(personID);

# Alter Command

1. ALTER Table Student ADD Address VARCHAR2(20);
2. ALTER Table Student MODIFY Address VARCHAR2(50);
3. ALTER TABLE Student RENAME COLUMN Address TO R_Add
4. ALTER Table Student DROP (R_Add)
5. ALTER Table student ADD CONSTRAINT STud-Sid-PK PRIMARY KEY (StudentID)
6. ALTER Table Student Drop CONSTRAINT Stud-sid-PK.

---

→ Drop one Column

ALTER TABLE Student DROP (DOB);

→ Drop Two Column

ALTER TABLE Student Drop (Gender, mobNO);

→ Add one Column

ALTER TABLE Student Add Address varchar2(20);

→ Add two Column

ALTER TABLE Student ADD (Course VARCHAR2(20), marks Number (10));

---

## Case

Query

```
Select Distinct Itemtype,
Case
    when price between 0 And 499 then 'cheap'
    When Price between 500 and 1999 then 'Affordable'.
    when price between 2000 And 4999 then 'Expensive'
    when price >= 500 then 'veryExpensive';
    End AS CLASSIFICATION.
from Item order By ItemType.
```

Syntax →

```
Case
    When ----
    when ----
    else ---
End
```

# DML Command

## Insert Statement Syntax

Table name

(1) Insert into Employee values (1, "Sachin", "mathura", "BCA");

(2) Insert into Employee (ID, Name, City) values (1, "Sachin", "mathura")

(3) Insert into Employee (ID, Name, City) SELECT QUERY

## Select Statement Syntax :-

(1) SELECT * From Employee // to See all tuple in table.

(2) Select Id, EName From Employee // to See only Id, Ename,

(3) Select Id, Ename from Employee Where Salary > 40000.

### Third — Where Clause

=> 1 Select Distinct Dept from Employee // print only distinct value.

=> Select ID, EName from Employee where Salary >= 3000 And DEPT = 'ETA';

=> " " " " " " OR " "

=> Select ID, EName from Employee where Salary BETWOGEN 3000 and 5000

### Third

(1) Select ID, Ename from Employee Where ID IN(2,3);
// It will print data of 2 and 3 Id.

(2) Select ID, Ename from Employee where ID Not IN(2,3);
// It will print all ID's data Exept (2 or 3)

(3) Select ID, Ename from Employee Where BONUS IS NULL

(4) " " " " " " IS NOT NULL

4rth

→ SELECT ID, Ename from Employee where ~~dept~~ dept ~~is~~ = 'pm'

→ Select ID, Name from Employee where dept = ' pm';

↳ // if in dat ' pm' is present then print

Its a leading Space → print accordingly.

→ trailing Space

→ varchar 2

↳ Select Id, name from Employee Where Ename = 'James Potter';

% Like

Ename

James Potter

Ethan mccarty

Emily Rayner

Jack Abraham

→ Select ID, EName from Employee where Ename LIKE 'E%'.

→ Select ID, Ename from Employee Where Ename Like '%r'

→ Select ID, Ename from Employee Where Ename Like '%m%'

↳ // Print which name hase m character.

→ Select ID, Ename from Employee Where DOJ Like '%14'

↳ // Print the date which have 14 in last (10-08-14)

→ Select ID, Ename from Employee Where DOJ like '__-__-__'

↳ print 10-08-14 fromatted date.

→ Seelect ID, Ename from Employee Where Ename Like '_a%'

↳ print which name hase a ~~in~~ char at Secord position.

## ⇒ Update Statement Syntax

(1) update Employee Set Salary = Salary * 1.1;
(2) update Employee Set Salary = Salary * 1.2 where Id = 1;
(3) update Employee Set Salary = Salary * 1.2, Bonus = 100 where Id = 1;

## ⇒ Delete Statement Syntax

(1) Delete from Employee
(2) Delete from Employee Where Dept = 'ETA'

## ⇒ Turncate Statement

(1) TRUNCATE TABLE Employee.

Note:
Turncate statement Can also be used to delete data from table. Turncate statemet delete all records from the table as it does not support Where clouse.

Turncate Statement is a faster option Compair to delete when you have to delete all records from the table.

# Functions

## ① Numeric Functions :-

1. ABS (10.27) ⇒ 10.27
2. ROUND (10.27) ⇒ 10
   ROUND (10.55) ⇒ 11
3. CEIL (10.27) ⇒ 11
4. FLOOR (10.78) ⇒ 10
5. ROUND (10.27, 1) ⇒ 10.3

↳ Round off only one digit.

**Queries : Ceil, floor and ABS**

- Select City, mintemp, CEIL(mintemp), floor(mintemp), ABS(mintemp) FROM Weather;

London | 10.27 | 11 | 10 | 10.27

# Conversion function

(1) To_char ⇒ To_char (value, format)

Exa - value := 10.78

To_CHAR (value, '9, 9.99') ⇒ 1, 0.78

'99.999' = 10.780

'9.9' = ####

(2) To_Number

To_Number (value, formate);

To_Number ('1000') ⇒ It Converts String to No.

⇒ To_CHAR - Date. date = 01-Jan-2019

To_CHAR (date, 'mon') ⇒ Jan (half month)

To_CHAR (date, 'month') ⇒ January (full month)

To_CHAR (date, 'dy') ⇒ wed (day)

To_CHAR (date, 'Day') ⇒ wednesday (full_day)

⇒ TO_CHAR - formate date.

To_CHAR (date, 'DD/mm/CCYY') = 01/01/2019

To_CHAR (date, 'DD/mm'YY') = 01/01/19.

Default date ⇒ 01-01-19

# Date function

Date
Page No:

(1) SYSDATE , SYSTIMESTAMP

08-DEC-21 08-DEC-21 05.56.21.787 AM

(2) ADD_MONTHS ⇒ ADD_MONTHS (date, n)

Eg = RecordDate = 01-Jan-14

// Adds n months to the given date.

⇒ ADD_MONTHS (RecordDate, 1) ⇒ 01-Feb-14

⇒ ADD_MONTHS (RecordDate, 3) ⇒ 01-Apr-14

increas months.

(3) MONTHS_BETWEEN

Eg = MONTHS_BETWEEN ('01-Feb-2014', '01-Jan-2014')

= 1

MONTHS_BETWEEN ('01-Feb-2014', 01-Mar-2014)

= -1

MONTHS_BETWEEN ('01-Feb-2014', '01-Jan-2014')

mar

= 2

01-10-10

Va Shubham

Date

Page No

# Function

(1) Character function :-

1. UPPER ⇒ UPPER (value)
2. LOWER ⇒ LOWER (value)
3. CONCAT ⇒ Concat (val1, val2) // val1 || val2
4. LENGTH ⇒ LENGTH (value)

val1 = JaVa  val2 = 101

output

1. JAVA
2. Java
3. JaVa101
4. 4

⇒ Concatination

1. Concat (val1, val2) // JaVa101
2. val1 || val2 // JaVa101
3. Concat ( Concat ('val1', ',') , val2)

// JaVa, 101

⇒ Substring ⇒

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| D | A | T | A | B | A | S | E |

output

Substr ('DATABASE', 5) ⇒ BASE
(1 2 3 4 5)

SUBSTR ('DATABASE', 3, 3) ⇒ TAB
(1 2 3 / 1 2 3)

# Aggregate function

- Sum (total)
- Avg (average)
- MIN (lowest value)
- MAX (highest value)

- COUNT (Number of rows)

**Note**

All aggregate function ignore NULL values except Count (*)

⇒ MIN, MAX, SUM, COUNT

Exa

Salary = 25,000, 50,000, 30000

MIN (Salary) = 25000
MAX (Salary) = 50000
Sum (Salary) = 105000
Count (Salary) = 3

Count (*) ⇒ all.

⇒ COUNT with Distinct means Duplicate Count once.

Exa ID = 1, 2, 1, 3, 2, 1

= COUNT (Distinct ID) ⇒ 3

⇒ Average.

AVG (Salary) = 85000

Order By Statement

## Order By

→ order by must be used to specify the column on which data has to be sorted and the sort order.

Ascending ord descending. Rows are sorted in ascending order if Sort order is not Specified. DESC should be used to Sort the rows in descending order.

ORDER BY clause must be the last clouse and can be used only is select statement. The sort order only applies to the query result.

① Seelect Id, Ename, Salary From Employee ORDER BY EName.

② Sedect Id, Ename, Salary from Employee ORDER BY Enam, Salary.

③ Select Id, Ename from Employee ORDER BY Ename DESC

④ Select Id, Ename from Employee ORDER BY Ename Asc, Salary DESC

⑤ Select Id, Ename from Employee ORDER BY 2, 3

Group By

Query: SELECT Dept, Sum(Salary) From Employee Group by Dept;

| DEPT | SALARY |
|---|---|
| ETA | 25000 |
| ETA | 90,000 |
| ETA | 30,000 |
| ICP | 50,000 |
| ICP | 75000 |
| IVS | 35,000 |

| DEPT | Sum(Salary) |
|---|---|
| ETA | 145000 |
| ICP | 125000 |
| IVS | 35000 |

# Order of Query

| F | J | W | G | H | S | D | O |
|---|---|---|---|---|---|---|---|
| FROM | JOIN | WHERE | GROUP BY | HAVING | SELECT | DISTINCT | ORDER BY |


* In order to get the details from multiple table or from the same table based on some condition in a single output table, SQL provid the concept of Unions and Joins.

(1) UNION : Removes all duplicates from the Result entire Columns,

```
SELECT * from Employee Where Degignation = 'pm'
UNION
SELECT * from Employee Where DEPT = 'ICP'.
```

Example

→ Select CompId from Employee UNION Select CompId from Computer.

| Employee |
|---|
| CompId |
| 1 |
| 2 |
| Null |

| Computer |
|---|
| CompId |
| Null |
| 1 |
| 2 |

Output

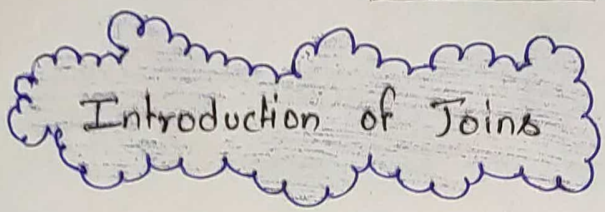
| CompId |
|---|
| 1 |
| 2 |
| Null |

(2) UNION ALL :

⇒ SELECT CompId from Employee UNION ALL SELECT CompId from Computer.

output :-

| CompId |
|---|
| 1 |
| 2 |
| NULL |
| NULL |
| 1 |
| 2 |




⇒ To display Employe details Along with Computer table details, in Single tabular formate.

⇒ The Sample output is provided below.

Employee table

| ID | Ename | CompId |
|---|---|---|
| 1 | James | 1001 |
| 2 | Eathan | NULL |

Computer table

| CompId | Model |
|---|---|
| 1001 | Vostro |
| 1002 | Precision |

Result table

| ID | Ename | CompId | model. |
|---|---|---|---|
| 1 | James | 1001 | vostro |

Joins are multiple types :-

1. Inner Join
2. Self Join
3. Left outer Join
4. Right outer Join
5. full outer Join.

(1) Cross Join

Cross Joins means Combination of one table's Item to Second table's item one by one.

Exq Employee table

| ID | NAME | CompID |
|---|---|---|
| 1 | Sachin | 1001 |
| 2 | Saurav | 1002 |
| 3 | mohit | 1003 |

Computer table.

| CompId | model |
|---|---|
| 1001 | vashro |
| 1002 | Precision |
| 1003 | Edge. |

⇒ Select E.ID, E.NAME, C.CompID, C.model
from Employee E Cross Join Computer C

| Id | Name | CompId | model |
|---|---|---|---|
| 1 | Sachin | 1001 | vashro |
| 2 | Saurav | 1001 | vashro |
| 3 | mohit | 1001 | vashro |
| 1 | Sachin | 1002 | precision |
| 2 | Saurav | 1002 | precision |
| 3 | mohit | 1002 | precision |
| 1 | Sachin | 1003 | Edge |
| 2 | Saurav | 1003 | Edge |
| 3 | mohit | 1003 | Edge |

(2) Inner Join

→ Inner join return only the matched Row.

→ QUERI

```
SELECT ID, NAME, E.COMPID, C.COMPID, MODEL FROM EMPLOYEE
    INNER JOIN COMPUTER C ON E.COMPID = C.COMPID
```

Result

| ID | Name | E_CompID | C_CompID | model |
|---|---|---|---|---|
| 2 | Saurav | 1002 | 1002 | Precision |
| 3 | Mohit | 1003 | 1003 | Edge. |

(3) Inner Self Join : When a table is joined with itself it is called Self join [Inner Join]

| ID | Ename | Salary | Bonus | DEPT | manager |
|---|---|---|---|---|---|
| 1 | James | 75000 | 1000 | ICP | NULL |
| 2 | Ethan | 90,000 | 12000 | ETA | NULL |
| 3 | Emily | 25,000 | 100 | ETA | 2 |
| 4 | Jack | 30,000 | NULL | ETA | 2 |
| 5 | Ayaz | 40,000 | NULL | ICP | 1 |

Query

→ SELECT Emp.ID EMPID , Emp.ENAME EMPNAME , mgr.ID MGRID, mgr.NAME MGRNAME. From Employee Emp Inner Join Employee mgr ON Emp.manager = mgr.ID

Output

| EmpID | EmpNAME | MGRID | MGRNAME |
|---|---|---|---|
| 3 | Emily | 2 | Ethan |
| 4 | Jack | 2 | Ethan |
| 5 | Ayaz | 1 | James |

for understand

Emp (child table)

| ID | Ename | manager |
|---|---|---|
| 1 | James | NULL |
| 2 | Ethan | NULL |
| 3 | Emily | 2 |
| 4 | Jack | 2 |
| 5 | Ayaz | 1 |

mgr (parent table)

| ID | Ename | manager |
|---|---|---|
| 1 | James | NULL |
| 2 | Ethan | NULL |
| 3 | Emily | 2 |
| 4 | Jack | 2 |
| 5 | Ayaz | 1 |

(3) left

# LEFT OUTER JOIN

Query:

```
SELECT ID, ENAME, E.COMPID AS E_COMPID, C.COMPID.C_COMPID
    MODEL from Employee E LEFT OUTER JOIN COMPUTER C
                ON E.COMPID = C.COMPID
```

Example

Emp TABLE

| ID | NAME | DEPT | COMPID |
|---|---|---|---|
| 1 | James | ICP | 1001 |
| 2 | Ethan | ETA | NULL |
| 3 | Emily | ETA | 1002 |

Computer table

| CompID | MAKE | model |
|---|---|---|
| 1001 | Dell | vostro |
| 1002 | Dell | Edge |
| 1003 | Lenovo | Horizon |

⇒ Select ID, NAME, E.compID, C.compID, model from Employee E left outer Join Computer C ON E.compID = C.compID.

Result

| ID | NAME | ECompID | CCompID | model |
|---|---|---|---|---|
| 1 | James | 1001 | 1001 | vostro |
| 2 | Ethan | ~~NULL~~ 1001 | NULL | NULL |
| 3 | Emily | 1002 | 1002 | Edge |

Note In left outer Join only left Side table of data will print and in Right outer join Right Side of table's data will print.

(4) Right outer join

Query:

```
Select ID, Name, E.CompID, C.CompID, model from Employee E
  Right outer join Computer C ON E.CompID = C.CompID
```

Result:

| ID | NAME | ECompID | CCompID | model |
|---|---|---|---|---|
| 1 | James | 1001 | 1001 | Vostro |
| 3 | Emily | 1002 | 1002 | Edge |
| NULL | NULL | NULL | 1003 | Horizon |

(5) Full outer Join

Query:

```
Select ID, Name, E.CompID, C.CompID, model from Employee E
Full outer Join Computer C on E.CompID = C.CompID
```

Result

| ID | Name | ECompID | CCompID | model. |
|---|---|---|---|---|
| 1 | James | ICP | 1001 | Vostro |
| 2 | Ethan | ETA | NULL | NULL |
| 3 | Emily | ETA | 1002 | Edge |
| NULL | NULL | NULL | 1003 | Horizon. |

mysql Don't allow this full outer join but we can do acceive it by left outer join and Right outer join

Query: Select ---- from Emp E left outer join Computer C on E.__ = C.__
UNION
Select -- from Emp E Right outer join Computer C on E.__ = C.__