

# Introduction of DBMS



Data :-

These are facts (values or figures) which can be recorded.

Data Base :- It is a collection of logically inter-related data. OR

It is a collection of information that is organized so that it can be easily accessed, managed and updated.

DBMS :-

It is a software system which facilitated defining, constructing, manipulating and sharing of data base among users and programs.

DBMS is a set of inter-related data and a collection of program to access that data.

Application of DBMS :-

Sector

Use of DBMS

(i) In Universities

- For student information, course registrations, fee collection, colleges and grades.



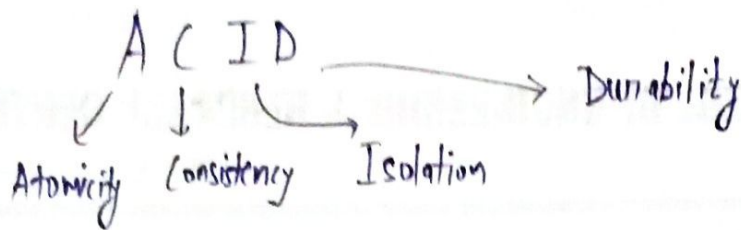
- (ii) Banking - For customer information, account activities, payments, deposits, loans etc.
- (iii) Airlines/Railways - Ticket reservation and schedule information
- (iv) Telecommunication - It helps to keep call records, monthly bills, maintaining balances etc.
- (v) Finance - Stock information, sales & purchase of stocks & bonds.
- (vi) Marketing - Sales, purchases, product & customer information
- (vii) Hospital - Patient information, medication sales, payment, doctors
- (viii) Social media - User information
- (ix) HR Management - For information about employees, salaries, payroll, deduction, generation of paychecks etc.

GOAL and PURPOSE of DBMS:  $\Rightarrow$

GOAL: -

Goal of DBMS is to effectively store the data in an organised way and provide means to retrieve/extract data quickly & efficiently.





Atomicity:- We mean that either the entire transaction takes place at once or doesn't happen at all. Transactions do not occur partially.

Operations:-

Abort, Commit

Consistency:- This means that integrity constraints must be maintained so that database is consistent before and after the transaction.

Eg- before  $T = 500 + 200 = 700$   
 After  $T = 400 + 300 = 700$

Isolation:- Multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. And changes should be visible only after they have been made to the main memory.

Durability:- Once the transaction has completed execution, the updates & modifications to the database are stored in and written to disk and they persist even if a system failure occurs.



## Characteristic / PURPOSE of DBMS:—

- (i) Easy to access ✓
- (ii) Provides security ✓
- (iii) Reduce Data redundancy ✓
- (iv) It follows the ACID concepts (Atomicity, Consistency, Isolation and Durability) ✓
- (v) Better Data integrity ✓
- (vi) Sharing of Data ✓
- (vii) Multi-user environment. ✓

## ★ FILE SYSTEM v/s DBMS:—

File processing system stores permanent data in traditional operating system file. This has certain problem.

FILE system	DBMS
(i) Difficulty in accessing the data	(i) Data can be accessed easily.
(ii) Various security issue	(ii) Prevents security problem and unauthorized access.
(iii) Data redundancy may lead to difficulty in management and higher cost.	(iii) Reduce data redundancy
(iv) Occurance of data inconsistency	(iv) Data remains consistence. (accessed data being available)
(v) Difficulty to implement constraints	(v) Better data integrity. (accessed data being correct)



- |        |  |        |  |
|--------|--|--------|--|
| (vi)   | Data modifications may or may not completely entirely. | (vi)   | Data modification will either completely / entirely or will roll back to previous state. (Atomicity) |
| (vii)  | Files are independent and scatters.                    | (vii)  | Files are inter-related (isolation)  |
| (viii) | Difficult to share and access data by multiple users.  | (viii) | Easy to share and access data. (Data-sharing)  |



### View of Data (Data Abstraction) $\Rightarrow$

It is the process in which details of data organisation and storage are hidden and only essential features are shown as per requirement to users.

This abstraction is divided into 3 levels:-

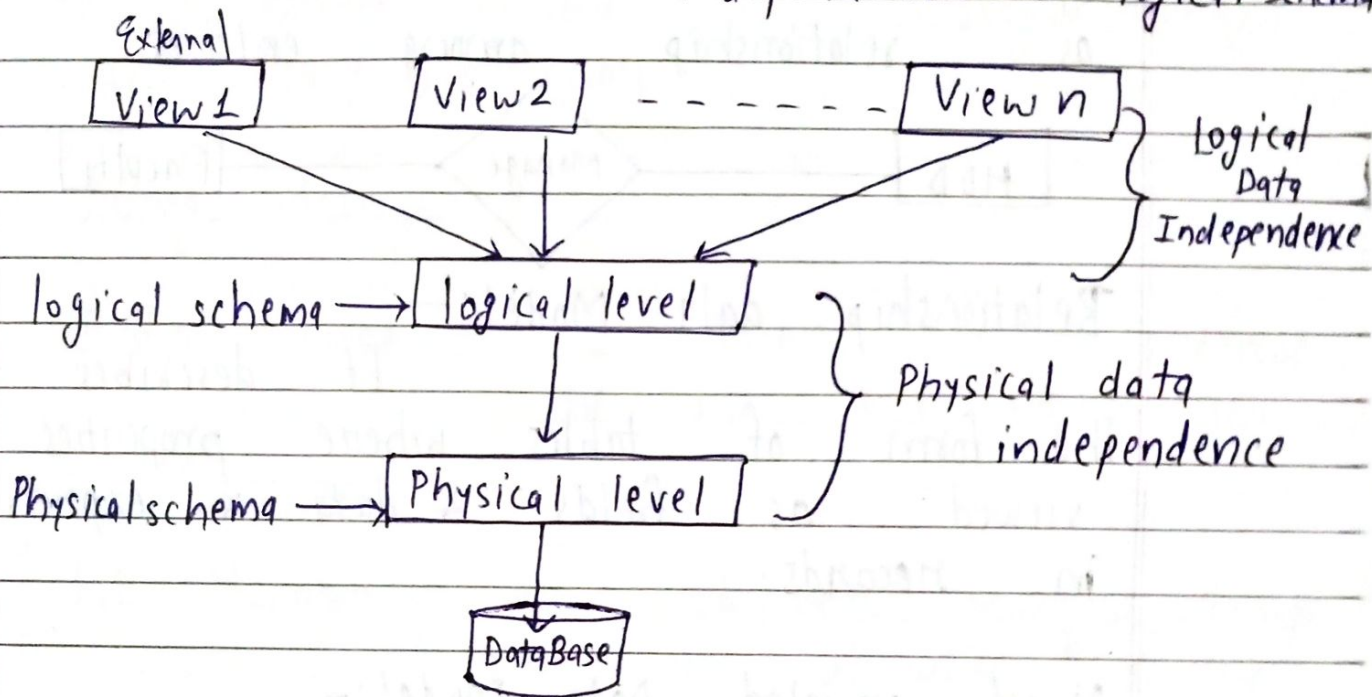
- (i) Physical level - It specifies how the data is actually stored. (centralized or distributed)
- (ii) logical level - It specifies what <sup>for</sup> data store, structure of DB and relationship b/w data.
- (iii) View level - It specifies only the part of the entire database which is currently viewed. There are different levels of views.

Instance  $\Rightarrow$  Database at any particular moment  
 Schema  $\Rightarrow$  It is the overall design of the database.



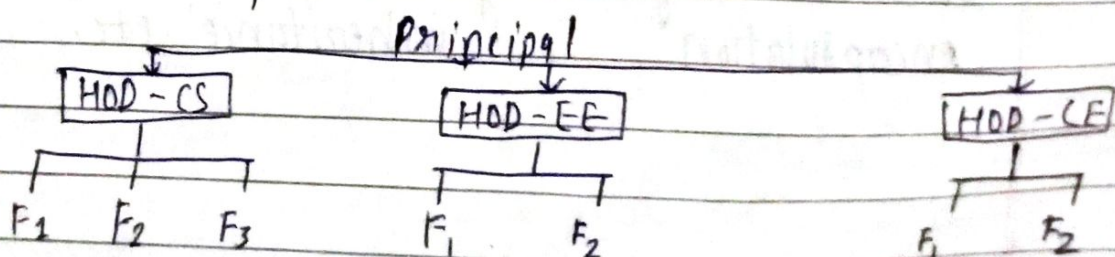
Physical data independence  $\Rightarrow$  When the logical level is not dependent on the physical schema of DBMS

Logical data independence  $\Rightarrow$  When view level is independent of logical schema.



Data Models  $\Rightarrow$  The overall structure of database is given by a data model. It provides tools to describe data and the relationship b/w data.

Hierarchical  $\Rightarrow$  This data model specifies data in a tree like structure. Traversing of data is done through pointer and only to directly link values.





Network  $\Rightarrow$  Data is described in a graph like structure. It is extremely complex.

ER data Models:-

It perceives / views data objects as entities & their association as relationship among entities.



Relationship data Model:-

It describes data in the form of tables where properties are viewed as fields & data is expressed in records.

Object oriented Data Model:-

It is the ER model with notion of object oriented programming such as objects, encapsulation, inheritance etc.

Object Relational Data model:-

It is the relational model with notion of object oriented programming such as objects, encapsulation, inheritance etc.



## \* Data Base structure $\Rightarrow$

Data Base structure or Architecture consists of various modules which are used to maintain the data base. It is responsible for storing, fetching, interfacing etc. the data.

It is broadly classified into storage manager and Query processor.

### (1) Storage Manager $\Rightarrow$

- (A) Buffer Manager:— It fetches data to and from the disk. And stores user queries.
- (B) File Manager:— This component manages storage space and allocation of disk.
- (C) Transaction Manager:— It ensures that the data remains in a consistent state.
- (d) Authentication & Integrity Manager:— This component validates and verifies user and queries.

All these components implement and maintain the data base in

- (i) Data file:— The actual data is contented by data file.



(ii) Data Dictionary :— It contains metadata about the data base.

(iii) Index :— For fast accessing the data.

(2) Query Processor  $\Rightarrow$

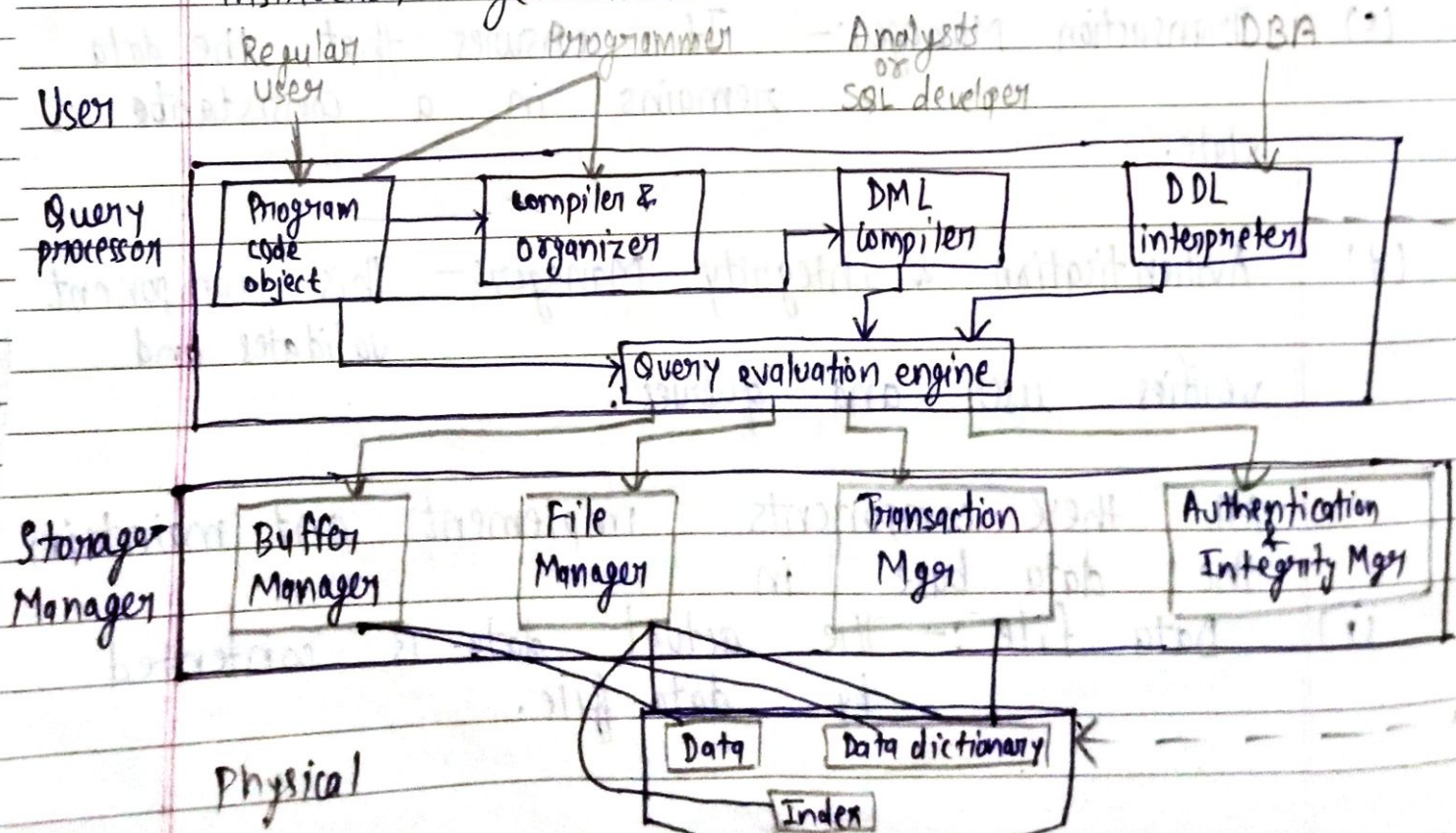
(a) DML compiler :— It translates DML commands either from user or program.

(b) DDL interpreter :—

It interpretes data definition commands and stores them in data dictionary.

Query evaluation Engine  $\Rightarrow$

It executes low level instruction generated.





## DataBase Design Process $\Rightarrow$

- (i) Requirement Analysis
- (ii) Conceptual design
- (iii) Logical design
- (iv) Schema refinement

(i) Requirement Analysis:- This step describes the type of data store and to whom the database is targeted for.

(ii) Conceptual Design:-

Based on the specification from previous step. A Layout can be prepared which describes the data from itself and association with other data.

(iii) Logical Design:-

After the concept to describe the data description is defined it can be implemented in a data model such as relational.

(iv) Schema refinement :-

It is the process of refining the relations to reduce redundancy.

## ER DATA MODEL $\Rightarrow$

It views real world objects as entity and model their association with other entities using relationship.



Entity set :-

Entity is a distinguishable object identified by observable property known as attribute. A collection of entity with similar attribute is called an entity set.

Tangible  
Intangible

Types of attribute :-

(i) Simple & Composite attribute  $\Rightarrow$

Simple attribute :- Attributes which are not breakable into sub-attributes are simple such as roll no. and the remaining ones are composite such as name (first name, last name)

(ii) Single valued & Multi valued attribute  $\Rightarrow$

When an attribute has a single value. It is single valued otherwise multivalued such as phone no., email id etc.

(iii) Derived Attribute :-

Attribute which can be obtained from other (stored) attribute such as age. eg - current age can be calculated by DOB, Google map location.

(iv) Descriptive Attribute :-

Properties of a relationship are descriptive attribute. Such as class timings b/w student & faculty.



Relationship set:-

Relationship is the association among entities and a set of similar relationship is a relationship set. And is given by  $R$  which is a subset of

$$\{ (e_1, e_2, e_3, \dots) \mid e_1 \in E_1, e_2 \in E_2, \dots \}$$

where  $e_1, e_2, \dots \rightarrow$  relationship

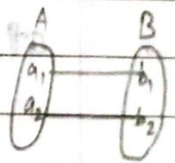
$E_1, E_2, \dots \rightarrow$  Entities.

\* Mapping Cardinalities :-

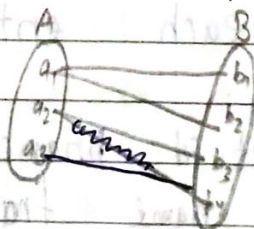
These are the no. of entities participating in a relationship. There are also known as Cardinality ratios.

(i) 1 to 1 Cardinality:-

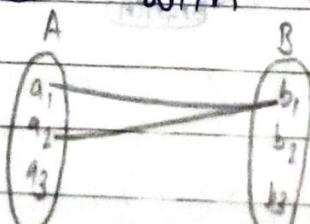
Atmost one entity in A is associated with one entity in B.

(ii) 1 to Many Cardinality:-

Atmost one entity in A is associated with multiple entity in B.

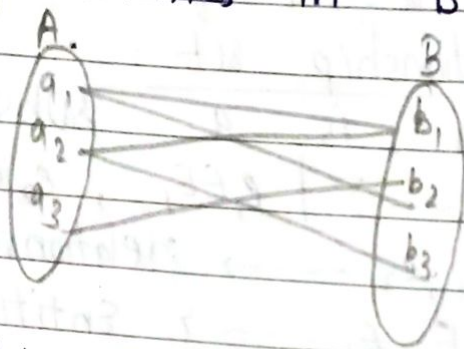
(iii) Many to 1 Cardinality:-

Multiple entities in A can be associated with atmost one entity in B.





(iv) Many to Many Cardinality :- Multiple entities in A are associated with many entities in B.



(v) Participation :-

Entities in entity set are involved in at least one relationship. If not it is partial.

★ keys ⇒

Keys on key attribute is a unique attribute which distinguishes each entity in an entity set. It uniquely identify the entity.

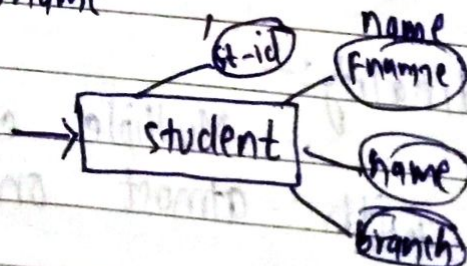
(a) Super key :-

It is a set of attributes which uniquely identifies each entity.

eg -

st-id name  
name fname

st-id branch, stid name branch  
frame branch





keys:- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships b/w tables.

Primary key:- It is the first key which is used to identify one & only one instance of an entity uniquely. (unique + not null)

Candidate key:- It is an attribute or set of an attribute which can uniquely identify a tuple. The remaining attribute except for primary key are considered as a candidate key.

Super key:- It is a combination of all possible attributes which can uniquely identify two tuples in a table.

Super set of any candidate key is superkey.

Foreign key:- It is an attribute or set of an attributes that references to primary key of same table or another table.



(b) **Candidate key** :— A super key may have extra attributes that may not be needed to identify an entity uniquely. Therefore we use candidate key. It is a set of attribute where no proper subset is a super key. Such minimum super key are candidate key.

eg -


st-id		sk	ck
st-id	name	sk	
st-id	branch	sk	
st-id	name	branch	sk
name	fname	sk	ck
name	fname	branch	sk

(c) **Primary key** :— It is an attribute selected by the database designer to identify the entity uniquely. It is selected from candidate key only.  $\{ \text{unique} + \text{not null} \}$



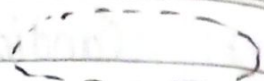

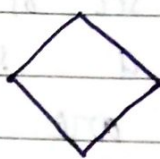


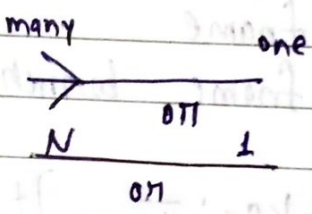
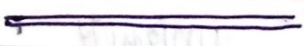
(d) **Foreign key** :— It is an attribute which serve as primary key in some other entity set.

### ★ ER Diagram ⇒

It is a graphical representation of the conceptual design of the data base structure. Its component include

(i) **Rectangle**  **entity set**



- (ii) Attribute  single valued  Multi valued  derived
- (iii) Flow 
- (iv) Relationship set 
- (v) Weak entity set 
- (vi) Weak Relationship set 
- (vii) Cardinality 
- (viii) Total participation 
- 

Weak Entity set :—

It is an Entity set which does not have a uniquely identifying attribute and is dependend on the strong entity set.



Q. Draw an ER Diagram to represent project development for Employees in various department.

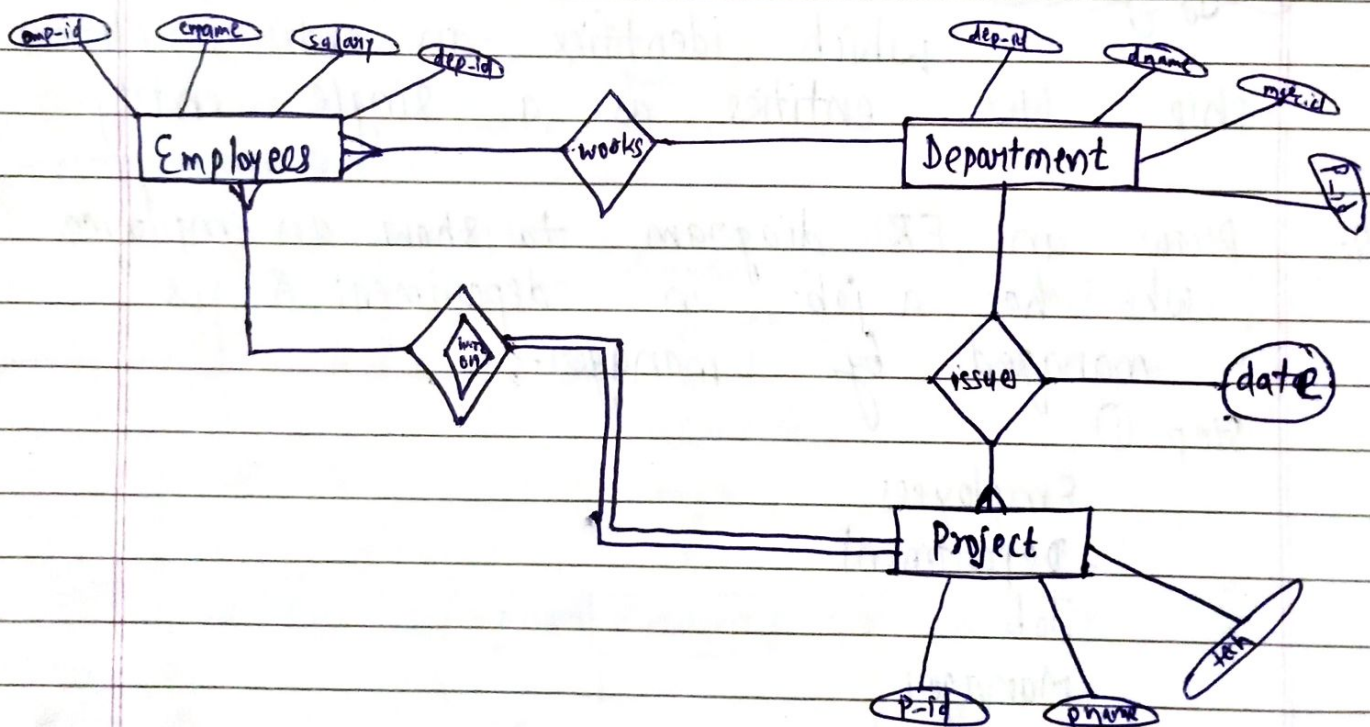
Ans - Step-① Entity Set :-

- Employees (emp-id, ename, salary, dep-id)
- Department (dep-id, dname, mgr-id, p-id)
- Project (p-id, pname, tech)

Step-②

works (Employees, department)

Issues (department, project) : date



Hierarchies ⇒

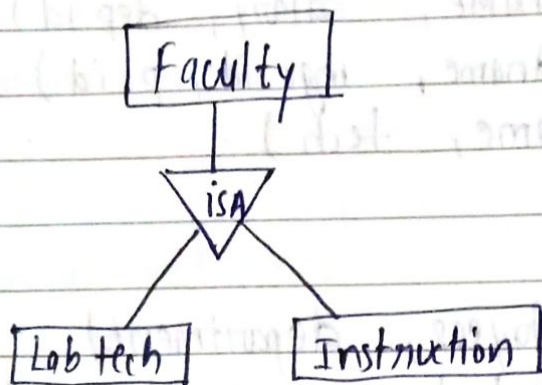
Generalization:- It is the synthesizing of entities to higher level entities based on common attributes.



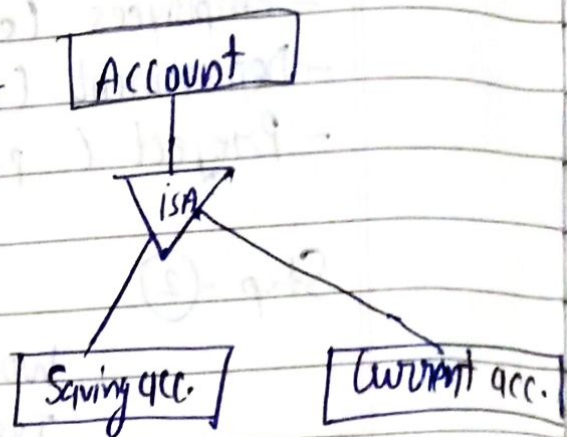
## Specialization :-

It is the refinement of entities into designated sub-grouping by identifying distinguishable properties.

eg -



eg -



Aggregation :- It is an abstraction process which identifies an entire relationship b/w entities as a single entity.

Q. Draw an ER diagram to show an employee who has a job in department & is managed by managers.

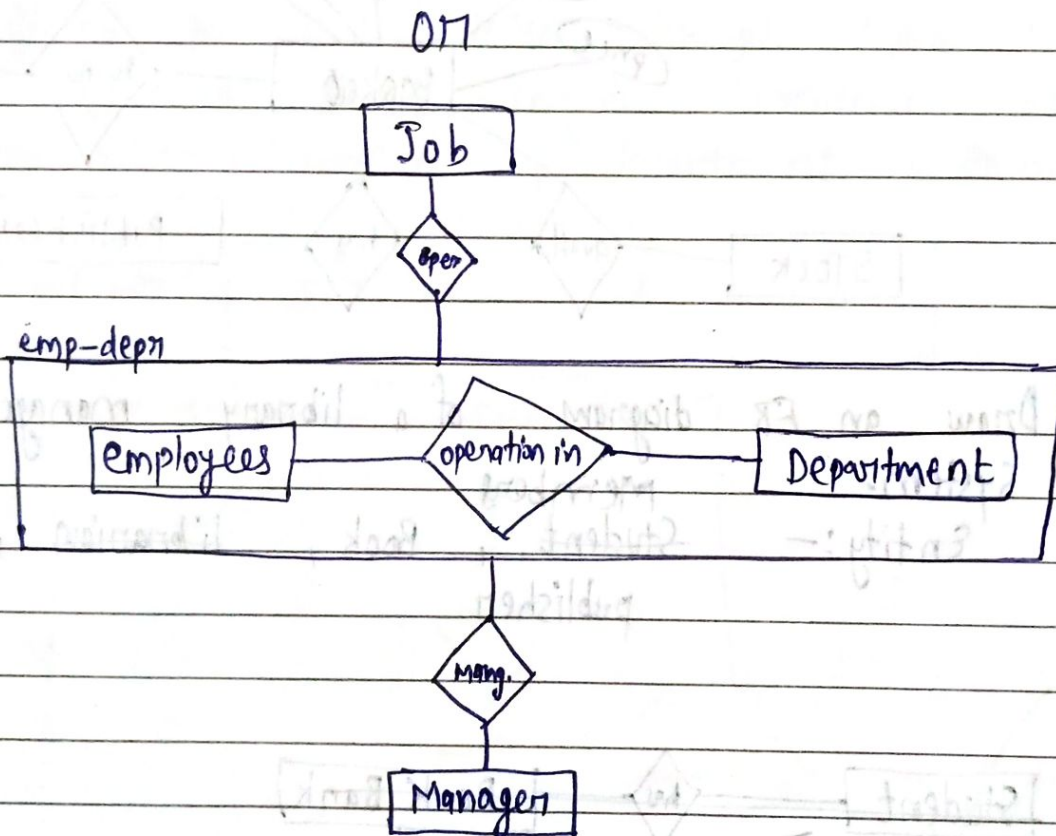
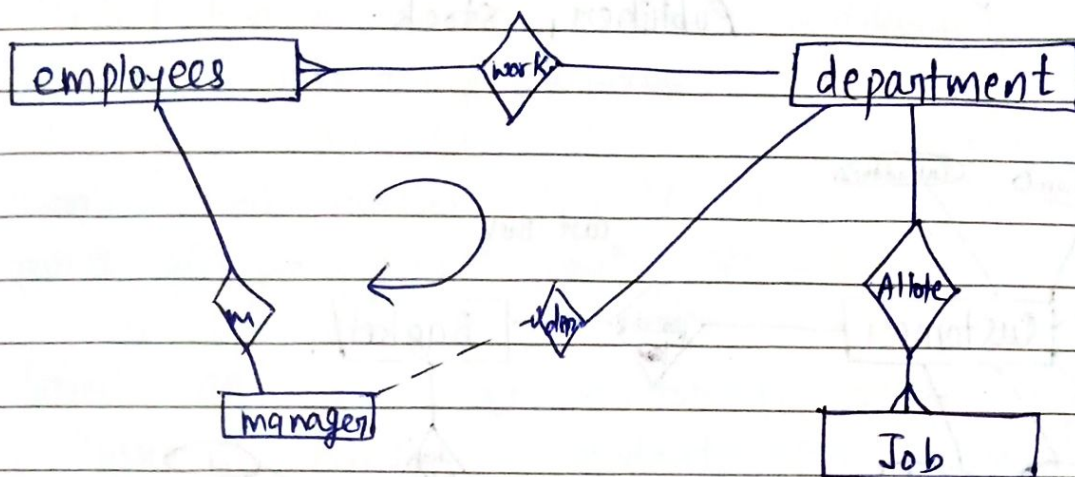
Ans - Step-①

Employees  
Department  
Job  
Manager

Step-②

works (Emp, depar.  
Allote (Depar,

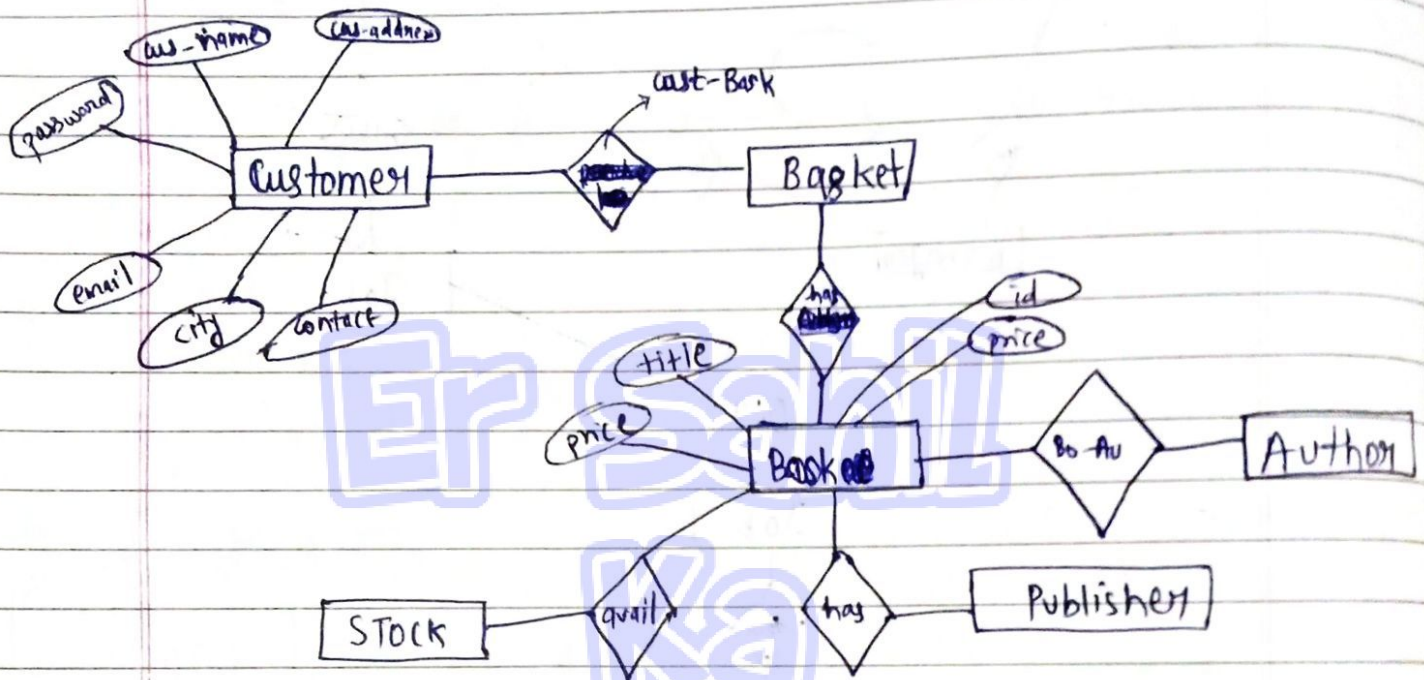




- Q. Draw an ER diagram for an online book store where customers can add their book in baskets and depending upon the availability purchased books can be either in a e-book or a hard copy. Books are identified by their author & publisher which may have also written other books.

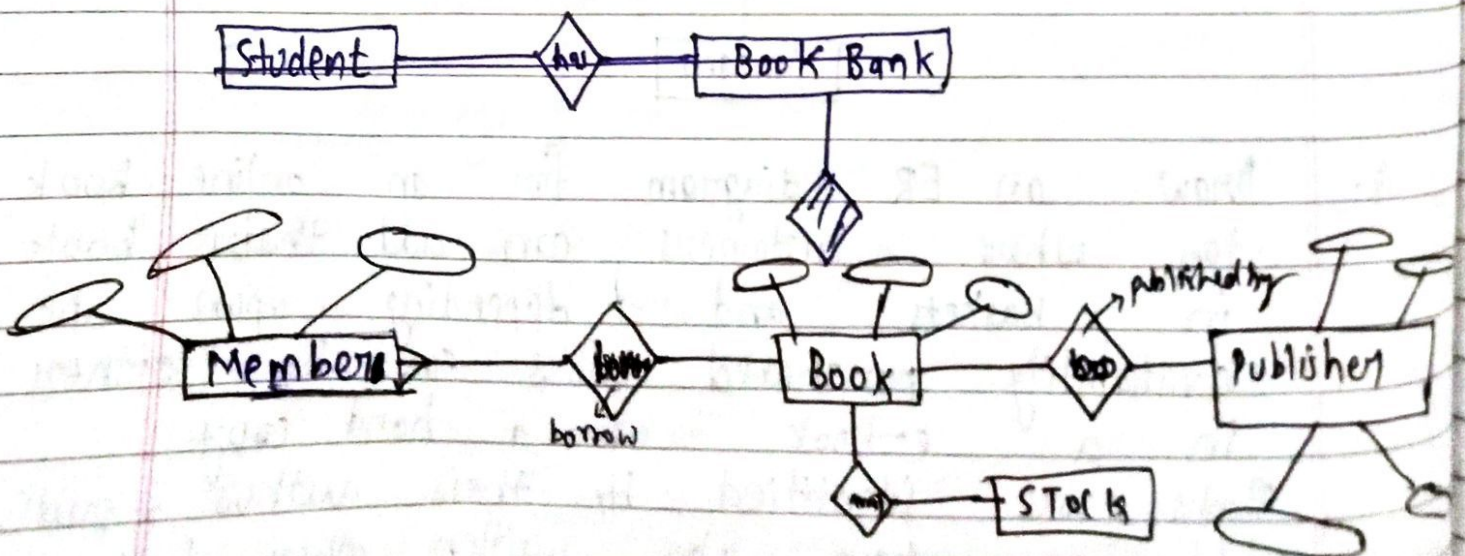


Q. Entity: — Basket, Book, customer, author, Publisher, stock.



Q. Draw an ER diagram of a library management system.

Ans. Entity: — Student, Book, librarian, Book Bank, publisher



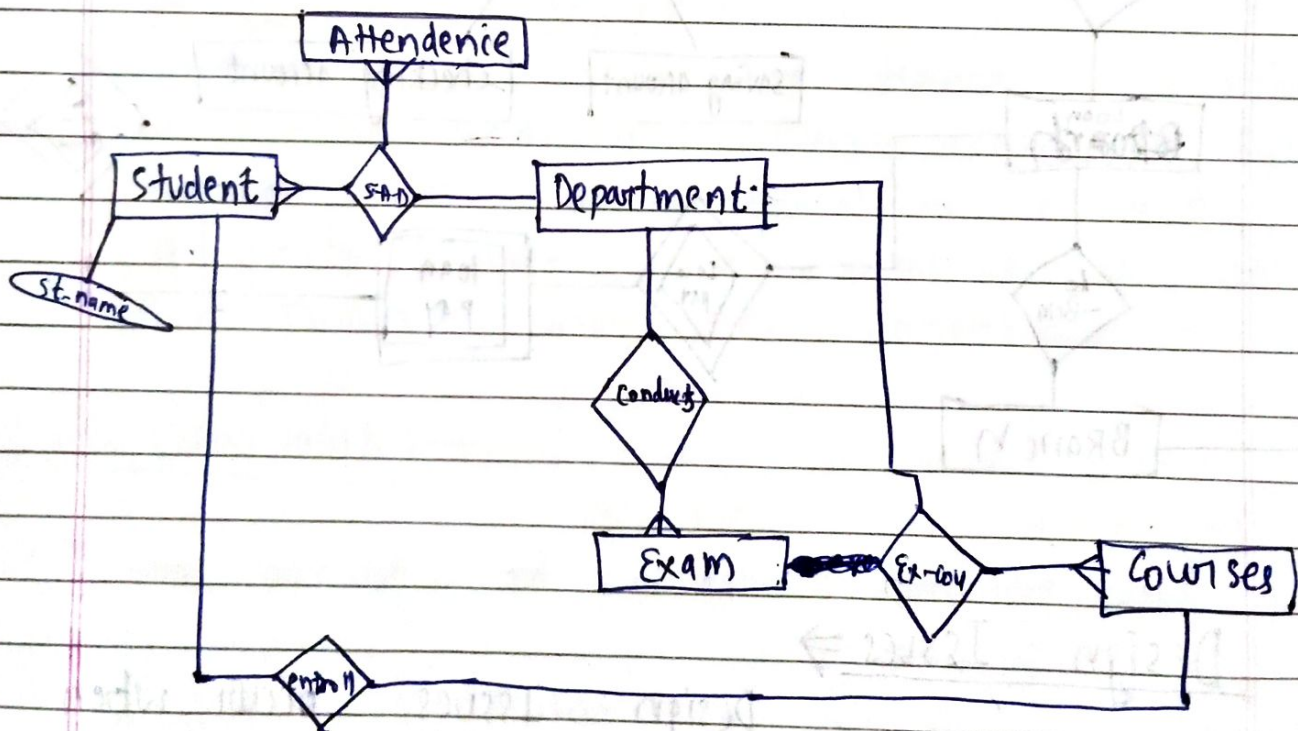


- ① Member ( m-id , name , book-id )  
book ( book-id , bname , subject , publisher-id )  
publisher ( publisher-id , pub-name )

- ② Borrow ( book , member ) : book-id  
published by ( book , pub ) : publisher-id

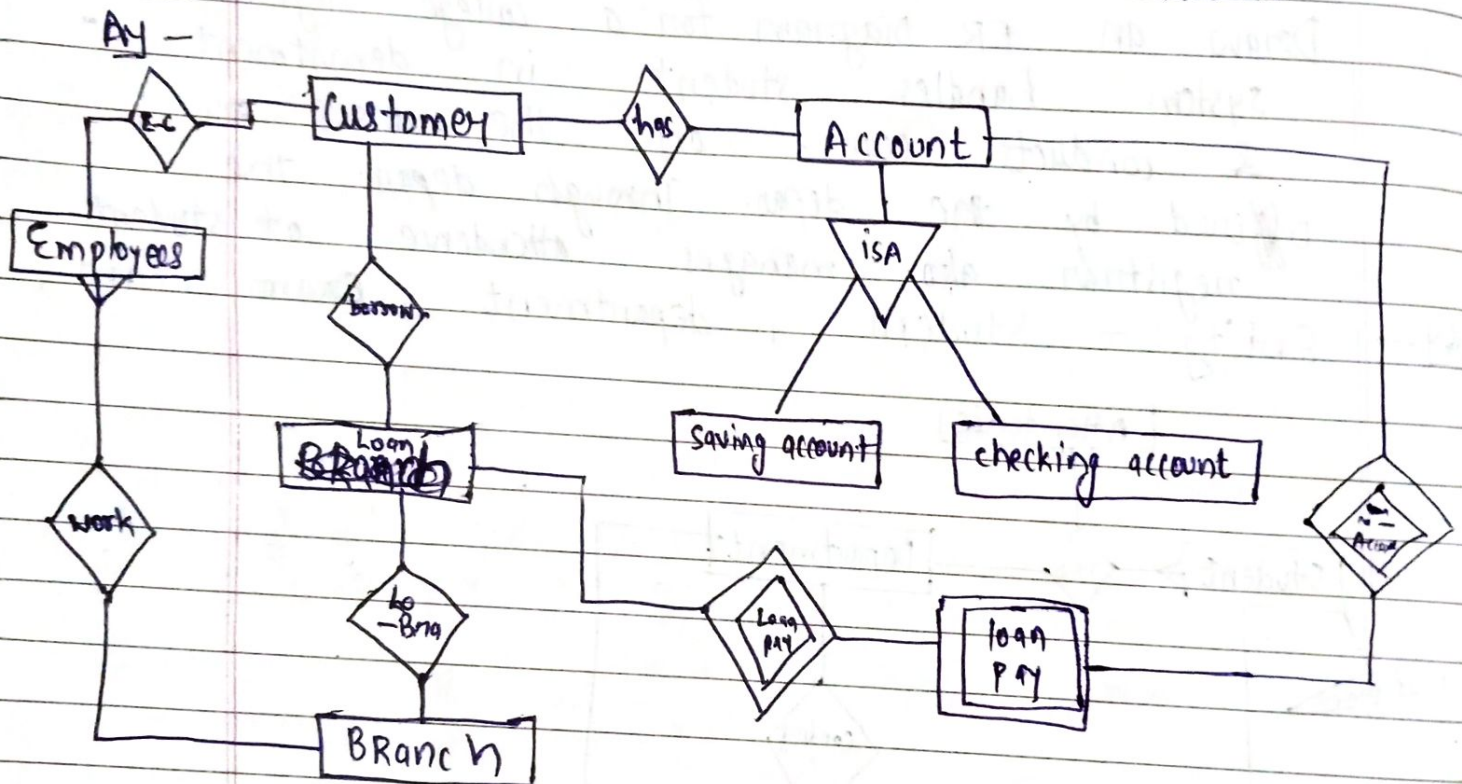
Q. Draw an ER-Diagram for a college registrar system handles student in department & conducts exam. on the courses offered by the depart. Through depart. the registrar also manages attendance of students.

dy- Entity: - Student , department , ~~Exam~~ Exam ,





Q. Draw an ER diagram of a Banking enterprise where customers can borrow loan. Customers can also operate a saving or a checking account through which it can pay loan amount. Every employees of this enterprises has also an account.



### Design Issues ⇒

Design Issues occur when it becomes difficult to differentiate b/w elements of ERD.

### Entity v/s Attribute ⇒

An object can be treated as entity or attribute depending upon the initial



requirement of user and whether the design requires to keep extra information. If as per requirement there is only one value for associating then it is an attribute and the relationship is removed otherwise it is an entity.

### Entity v/s Relationship $\Rightarrow$

An object can also behave as a relationship if user requirement do not necessitate descriptive attribute otherwise it may be treated as an entity.

### Binary vs n-ary $\Rightarrow$

Any ternary relationship can be expressed in Binary as well but when binary is converted to a ternary one, information may be specialised or additional information may be inserted.

### \* Constraints :-

These are rules implemented on the database to ensure correctness of data

#### $\rightarrow$ Integrity constraints:-

[not null, unique]

These statements allow the data to be a legal entity.

#### $\rightarrow$ Domain Constraints:-

[check]

These constraints ensure that data must satisfy a condition.



→ key constraints [Primary key]:—

These statements requires the subset of field to be key attributes.

→ Referential constraints [Foreign key]:—

For given two tables this constraints ensure that if the data is modified in one table consistency remains in another table as well.



## Unit - 2

# Relational Algebra & Calculus

### Relational Algebra $\Rightarrow$

It is a formal query language which is procedural in nature and inputs a sets of relations and after operation produces a new relation as result.

### OPERATION $\Rightarrow$

(a) PROJECTION ( $\pi$ ) : —

Returns list of fields (column) from relation (table).

(b) SELECT : — It is represented by  $\sigma$ . Returns list of tuples (rows) from table.

$\pi_{\text{projection}} (\text{Relation})$

$\sigma_{\text{selection}} (\text{Relation})$

$\pi_{\text{projection}} (\sigma_{\text{selection}} (\text{Relation}))$

→ select

from

→ where

eg- Display name & percentage of all students.

$\pi_{\text{projection}} \pi_{\text{sname, percent}} (\text{student})$

eg- Display name of cs students.

$\pi_{\text{sname}} (\sigma_{\text{branch}='cs'} (\text{student}))$



$\wedge \rightarrow$  and

DATE:  $\vee \rightarrow$  or /

PAGE NO.:  $\neg \rightarrow$  Not

eg - Display name, percent of 4 sem cs.  
Ans -

$\Pi$  name, percent ( $\sigma_{\text{branch} = 'cs' \wedge \text{sem} = '4'} (\text{Student})$ )

eg - Display all students except from cs branch.

Ans -  $\sigma_{\text{branch} \neq 'cs'} (\text{Student})$

eg - Display id & branch of student from 4 & 5 sem.  
Ans -

$\Pi$  id, branch ( $\sigma_{\text{sem} = '4' \vee \text{sem} = '5'} (\text{Student})$ )

eg - Display name of all the student who has scored b/w 70 to 80% and belong to civil branch.

Ans -  $\Pi$  name ( $\sigma_{\text{branch} = 'civil' \wedge 70 < \text{per} \wedge \text{per} < 80} (\text{Student})$ )

(c) Union ( $\cup$ )  $\Rightarrow$  It is a binary operation which is used to return information from one or more relation that are union compatible. The result will exist in either of the relation or both.

(d) Set intersection ( $\cap$ )  $\Rightarrow$  It is used to return common information from union compatible relations.



(e) Set difference  $(-)$   $\Rightarrow$  It returns information from ~~was~~ first relation that donot exist in the second one.

Sample :-

student ( sid , firstn , lastn , email )  
 results ( sid , cat , eno. , marks )  
 exercises ( cat , eno. , subject , maxmarks )  
 guests ( firstn , lastn , email )

Q. Display id & name of all students.

Ans -

$\pi$  sid , firstn , lastn ( student )

Q. Display id & category of students scoring less than 50 marks.

Ans -

$\pi$  sid , cat (  $\sigma$  marks < 50 (  $\pi$  result student ) )

Q. Display all the names of student that are permanent or temporary.

Ans -

~~Q. Name~~

$\pi$  firstn , lastn ( student )  $\cup$   $\pi$  firstn , lastn ( guests )

Q. Display names of all the students who are both guest & permanent.

Ans -

$\pi$  firstn , lastn ( student )  $\cap$   $\pi$  firstn , lastn ( guests )

Q. Display all names the permanent student.

Ans -

$\pi$  firstn , lastn ( student ) -  $\pi$  firstn , lastn ( guests )



$A \leftarrow \text{Tuition, section (students)}$   
 $B \leftarrow \text{Tuition, h (guest)}$   
 $A \cup B, A \cap B, A - B$

Rename ( $P$ )  $\Rightarrow$

$P_x(E)$   
 or  
 $P_x(A_1, A_2, \dots)(E)$

★ Joins  $\Rightarrow$

It is a concept which is used to associate two or more relations.

(a) cross join  $\Rightarrow$  Also called cross product returns multiple rows from a table associated with each row of first table. It is denoted by  $\times$ . It returns table 1 ~~into~~  $\times$  table 2 rows.

(b) Inner join  $\Rightarrow$   
(I) Equi join:-

It is a cross join with a condition specified by equal to ( $=$ ).

Q. Display name & department name of students.

Ans-

$\pi_{sname, dname} (\sigma_{\text{student\_did} = \text{dept\_did}} (\text{student} \times \text{dept}))$

Student (s\_id<sup>pk</sup>, sname, did)  $\rightarrow$  foreign key  
 dept (did<sup>pk</sup>, dname)



table - relation

rows - tuple

column - fields / attributes

DATE :

PAGE NO. :

091  
TLsname, dname ( ~~Student~~  $\bowtie$  Student  $\bowtie$  dept )  
↑  
student.did = dept.did

Q. Display name & department of each student.

(II) Theta join ( $\bowtie_{\theta}$ )  $\Rightarrow$  It combines two or more relation based on the condition Theta whose operator can be ( $<, >, =, <=, <>$ ).

(III) Natural join  $\Rightarrow$  It internally joins two or more relations based on common attributes.

Eg- TLsname, dname ( Student  $\bowtie$  dept )

(C) Outer join  $\Rightarrow$

Including the result produced by inner join. Outer join produces the result which may not satisfy the condition given the incompleteness of the ~~permission~~ information.

(I) Left join :- Produces <sup>extra</sup> results from the left side table ( $\bowtie_{\leftarrow}$ )

(II) Right join :- ( $\bowtie_{\rightarrow}$ ) :- Produces extra results from the right side table.

(III) Full join ( $\bowtie_{\text{full}}$ ) :- Produces extra results from both side table.



Q. Display the name of all student whether they have dept or not.

Ans -

TLsname, dname (student  $\bowtie$  dept)

eg -

A	XY
B	YZ
C	MN
D	
E	

Q. TLsname, dname (student  $\bowtie$  dept)

eg -

A	XY
B	YZ
C	MN
	OP
	ZY

Q. TLsname, dname (student  $\bowtie$  dept)

eg -

A	XY
B	YZ
C	MN
	OP
	ZY
D	
E	

Q. Write algebraic expression for the following schema.

branch (bname, city, assets)

account (ano, bname, bal)

customer (cname, ccity, cstreet)

loan (lno, bname, amt)

deposit (cname, ano)

borrow (cname, lno)

(i) Select loan information where branch is kukas



A1 -

 ~~$\Pi_{cname}$~~   $\sigma_{bname='kukas'} (loan)$ 

(ii) Show all customers from jaipur.

 $\Pi_{cname} (\sigma_{ccity='jaipur'} (customer))$ 

(iii) Find customers who have an account or taken a loan or both.

A1 -

 ~~$\Pi_{cname} (customer \times account)$~~ 
 ~~$\Pi_{cname} (\sigma_{customer.cname=account.cno})$~~ 
 ~~$\Pi_{bname} (account) \cup \Pi_{bname} (loan)$~~ 
 $\Pi_{cname} (deposit) \cup \Pi_{cname} (borrow)$ 

(iv) Find all the customers who have taken loan from branch.

A1 -

 $\Pi_{cname} (\sigma_{branch='kukas'} (loan \bowtie borrow))$ 

OR

 $A \leftarrow \rho_{A1} (loan) \bowtie_{A1.lno = B1.lno} \rho_{B1} (borrow)$ 
 $\Pi_{cname} \sigma_{bname='kukas'} (A)$ 

(v) Find branch name &amp; customer name of all customers who have an account. But do not have a loan.

A1 -  $\Pi_{bname, cname} (account \bowtie deposit)$



## ★ Relational Calculus $\Rightarrow$

It is a non-procedurable query language which is declarative in nature. It is of two types: —

- (i) Tuple Relational Calculus (TRC)
- (ii) Domain Relational Calculus (DRC)

### (i) TRC $\Rightarrow$

It is specified by Tuple variables which are either bound or free.

It is of the form  $\{ t \mid Q(t) \}$

Here  $t$  is the free tuple variable extracted from a relation  $R$  specified by the condition  $Q(t)$

Variable which are quantified are given by bound variables.

There exists two quantifiers

(a)  $\forall \rightarrow$  For all

(b)  $\exists \rightarrow$  Existential

Q. Display all information of customers.  
Ans —  $\{ t \mid t \in \text{customer} \}$

Q. Display account information of all customers.  
Ans —  $\{ t \mid t \in \text{account} \}$





Q. Display all information of account ~~their~~ where balance is above 10,000.

Ans -

$\{ t \mid \exists s \in \text{account}, t \in \text{account} \wedge s[\text{balance}] > 10000 \}$   
or

$\{ t \mid t, \exists s \in \text{account} \wedge s[\text{balance}] > 10000 \}$

Q. Display all information where loan amount is 5000.

Ans -

$\{ t \mid t, \exists s \in \text{loan} \wedge s[\text{amt}] = 5000 \}$

Q. Display names of all customers.

Ans

$\{ t \mid t, \exists s \in \text{customer}, t[\text{cname}] = s[\text{cname}] \}$

Q. Display name of all customer & city as well.

Ans -

$\{ t \mid t, \exists s \in \text{customer}, (t[\text{cname}] = s[\text{cname}]), (t[\text{ccity}] = s[\text{ccity}]) \}$

Q. Display all the branches having loan above 50000.

Ans -  $\{ t \mid (t, \exists s \in \text{loan} \wedge s[\text{amt}] > 50000), (t[\text{bname}] = s[\text{bname}]) \}$

Q. Display customer names along with their amt who have taken a loan.

Ans -  $\{ t \mid t, \exists s \in \text{loan}, \exists p \in \text{borrow}, (t[\text{cname}] = p[\text{cname}]) \wedge (t[\text{amt}] = s[\text{amt}]) \wedge (s[\text{lno}] = p[\text{lno}]) \}$



Q. Find all the customers who have an account or have taken a loan or both.

Ans -

$$\{ t \mid t \in \text{customer}, \exists s \in \text{deposit}, \exists p \in \text{borrow}, (t[\text{cname}] = s[\text{cname}]) \vee (t[\text{cname}] = p[\text{cname}]) \}$$

(ii) DRC (Domain Relational Calculus)  $\Rightarrow$

This calculus is specified on domain variables and is of the form

$$\{ (x_1, x_2, x_3, \dots) \mid p(x_1, x_2, \dots) \}$$

Here  $x_1, x_2, \dots$  represent attribute of a relation and  $p$  is the formula.

Q. Display names of all the students getting above 50 marks.

Ans -

students (sid, sname, sem, branch, marks)  
 $\{ (t) \mid \exists s, e, n, a \wedge a > 50 (\langle s, t, e, n, a \rangle \in \text{students}) \}$

Q. Display names of all department students (sid, sname, sem, branch, marks, dept)  
 dept (did, dname)

Ans -

~~$\{ (n) \mid \exists s, e, n, a, d (\langle s, t, e, n, a, d \rangle \in \text{students}) \}$~~

$\{ (n) \mid \exists d (\langle n, d \rangle \in \text{dept}) \}$



Q. Display student name & their department name.

Ans-

$$\{ (n, t) \mid \exists (s, e, n, a, z) (\langle s, t, e, n, a, z \rangle \in \text{student} \wedge \exists (d) \langle d, n \rangle \in \text{dept} \wedge z = d) \}$$

Q. Display name & sem of CS students.

Ans-

$$\{ (t, e) \mid \exists (s, a, n, z) \langle s, t, e, n, a, z \rangle \in \text{student} \wedge \exists (d, n) \langle d, n \rangle \in \text{dept} \wedge n = 'cs' \}$$

$$\text{or}$$

$$\{ (t, e) \mid \exists (s, a, n, z) \langle s, t, e, n, a, z \rangle \in \text{student} \wedge n = 'cs' \}$$

### Safety of Expressions $\Rightarrow$

Consider the following expressions

$$\{ t \mid (t \in \text{student}) \}$$

$$\{ (s, t, e, n, a, z) \mid \langle s, t, e, n, a, z \rangle \in \text{student} \}$$

If above queries are negated the result will display all information except students which can not measured therefore to restrict such expression domain ( $\text{dom}(p)$ ) is given. which implement restriction to the range of values upon which the condition is implemented. For DRC quantifiers provide such restrictions. Therefore the above expressions become



$\text{dom } \{ t \mid \neg (t \in \text{student}) \}$

Expressive Power  $\Rightarrow$

Relational algebra & relational calculus under restricted expressions have expressive power therefore all expressions expressed in algebra can be written in TRC & DRC and vice-versa.

Relational Algebra

Relational Calculus

- |                                     |                              |
|-------------------------------------|------------------------------|
| (i) It is procedural.               | (i) It is not procedural.    |
| (ii) It is prescriptive             | (ii) It is declarative.      |
| (iii) Sequence of operation matter. | (iii) Does not matter.       |
| (iv) It is domain independent       | (iv) It is domain dependent. |

BASIC FORM OF SQL  $\Rightarrow$

SQL  $\Rightarrow$  (Structured Query Language)

It is a 4th generation declarative language which queries the database.

Components of SQL  $\Rightarrow$



SOL  $\Rightarrow$

SOL stands for Structured Query language.

SOL is used to communicate with a database.

SOL is a domain-specific language used in programming & designed for managing data held in R-DBMS. SOL is case-insensitive language.

### Categories of SOL commands

(i) DDL (Data definition language)  $\Rightarrow$

DDL changes the structure of table like creating a table, deleting a table, altering a table etc.

• Some commands that come under DDL -

• Create • Alter • Drop • Truncate

(ii) Data Manipulation language (DML)  $\Rightarrow$

It is used to modify the database. It is responsible for all form of changes in database.

Some commands that come under DML -

• insert • update • delete.



(iii) DQL/DSL (Data Selection Language):-

DQL is used to fetch the data from the database.

It uses only one command:-

- SELECT

(iv) DCL (Data control language):- DCL commands are used to grant and take back authority from any database user.

Some commands that come under DCL:-

- Grant
- Revoke

(v) TCL (Transaction control language):- TCL commands can only use with DML commands only.

Some commands that come under TCL:-

- Commit
- Rollback
- savepoint

DBA users  $\Rightarrow$

$\Rightarrow$  SYS

$\Rightarrow$  SYSTEM

$\Rightarrow$  HR



- (i) DDL definitions
- (ii) DML commands
- (iii) Transaction controls
- (iv) View definitions
- (v) Nested queries
- (vi) Correlated queries
- (vii) DCL
- (viii) Joins

(i) DDL definitions  $\Rightarrow$

It is used to create and modify database schemas and other objects. It has the following commands:—

- (a) Create
- (b) alter
- (c) drop
- (d) truncate

(a) Q. Create a sample loan management database consisting of following table:—

- (i) branch ( bname, city, assets )
- (ii) account ( ano, bname )

Ans—

```
create table branch (
  bname varchar2(10) primary key,
  city   varchar2(20) unique,
  assets number(10,2) check ( assets > 0 );
```

```
create table account (
  ano number(10) primary key,
  bname varchar2(20) references branch ( bname );
```



(b) alter: —

alter table tname

- add
- drop column
- modify
- rename column
- rename to

Q. (a) Add column balance to account table.

(b) Make balance not null

(c) Change balance to funds.

Ans -

(a) alter table account add balance number(10,2) ;  
check (balance)

(c) alter table account rename column balance to funds ;

(b) alter table account modify balance not null ;

(c) Drop: —

drop table tname ;

(d) Truncate: — truncate table tname ;

(iii) DML definitions ⇒

manipulation operation It is used to perform on the data.

(a) Insert

(b) Update

(c) delete



(a) Insert:-

insert into tname values (data1, data2, ...);

eg- student (Rno, name, dob, marks)

insert into student values (15, 'sahil', '12-JAN-20', 80)

for skip data (15, null, ...)

(b) Update:-update ~~table~~ tname set  
col = newvalue  
where condition;

Q. Update roll no. of 15 to 25.

Ans-

update ~~table~~ student set  
Rno = 25  
where Rno = 15;(c) Delete:-

delete from tname where condition;

Q. Remove all students scoring less than 40 marks.

Ans-

delete from students  
where marks < 40;



(iii) DQL (Data Query language) :—

Basic Form  $\left[ \begin{array}{l} \text{select} \quad * \mid \text{colname} \mid \text{math op} \\ \text{from} \quad \text{tables} \\ \text{where} \quad \text{conditions} \\ \text{order by} \quad \text{colname}; \end{array} \right.$

Q. Student ( rno , name, sem, dno, percent )  
 dept ( dno , dname , lno , HOD )  
 location ( lno , lname )

(a) Display all the students.  
 Ans -

select \* from student;

(b) Display all the department name in this college.  
 Ans -

select dname from dept;

(c) Identify all the semester of the students.  
 Ans -

select distinct sem from student

(d) Display all the unique location in college.  
 Ans -

select distinct lname from location.

(e) Display name & percent of all students with an additional 5 % bonus.



Q1 - select name, percent ~~5~~ from student;

(f) Display name of all the student from 2nd sem.

Ans - select name  
from student  
where sem = 2;

(g) Display rollno & name of all the student from 4th sem who have ~~scored~~ scored above 60%.

Ans - Select rollno, name  
from student  
where sem = 4 and percent > 60;

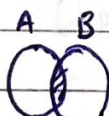
Set operators :-

Union



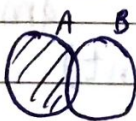
$Q_1 \cup Q_2$

Intersect



$Q_1 \cap Q_2$

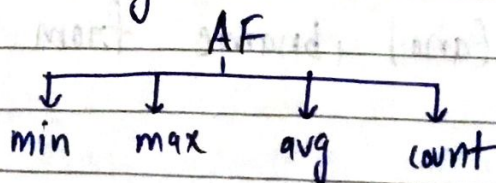
Minus



$Q_1 - Q_2$

Aggregate / Group functions :-

These are the functions which operate on multiple value in a column to produce a single value result.





Q. branch ( bname, city, assets)  
account ( ano, bname, bal)  
cust ( cname, city, street)  
loan ( lno, bname, amt)  
deposit ( cname, ano)  
borrow ( cname, lno)

(a) Find name of customers who have both loan & account

Ans -

select cname from deposit intersect select cname from borrow;

(b) Find all customers who have a loan but don't have a account.

Ans -

select cname from borrow minus select cname from deposit;

(c) Display maximum balance from all account.

Ans -

select max(bal) from account;

(d) Find the lowest amount for the loan

Ans -

select min(amt) from loan;

(e) Find the lowest amount for all branches.

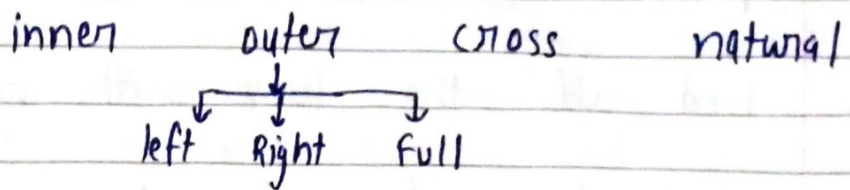
Ans -

select min(amt), bname from loan group by bname;

(f) Display no. of accounts for each branch.

Ans - select count(ano), bname from account group by bname;



Joins:—

Syntax →

```

select columnname
from table1 t jointype table2 u
on t.common = u.common
where condn
group by
order by
  
```

Q. Display balance of each branch with city there in.

Ans select b.bname, city, bal from branch b join  
account a on b.bname = a.bname.

Q. Find all customer name who have taken loan from the kukas branch worth more than 15000.

Ans - select cname from loan l join borrow b on  
l.lno = b.lno  
where amt > 15000 and bname = 'kukas';

Q. Find no. of accounts from jaipur.

Ans - select count (ano) from deposit d join cust c on  
c.cname = d.cname where ccity = 'Jaipur';

Q. Display all city of loan which exists b/w 100 and 500.

Ans - select ccity from cust c join borrow b on  
c.cname = b.cname  
where lno > 100 and lno < 500;



## Nested SQL OR SUB-QUERY $\Rightarrow$

Q. Find all the loans with no amount.

```
select lno. from loan
where amt is null;
```

$\Rightarrow$  syntax: — select  
from  
where col operation (select  
from  
where — );

Q. Find all the customers who have both loan and account.

Ans-

```
select d.cname from deposit intersect select cname
from bor;
```

```
select d.cname from dep d join bor b
on d.cname = b.cname
```

```
or
select cname
from dep natural join bor;
```

Q. Find all the customers who have an account in kukas branch but their bal. is below 5000.



Qy -

```
select cname from dep d join account a
on a.ano = d.ano
where bname = 'kukas' and bal < 5000;
```

OR

```
select cname
from dep
where ano in (select ano
from account
where bal < 5000 and bname = 'kukas');
```

Q. Display all the customer name who have borrowed loan from kukas branch.

Ans -

```
select cname
from borr
where branch lno in (select branch lno
from branch loan
where bname = 'kukas');
```

Q. Display all the customer who have account in jaipur.

Ans -

```
select cname
from dep
where ano in (select ano
from account
where bname in (select bname
from branch
where city = 'Jaipur'));
```



on

```
select cname
from branch b join account a
on b.bname = a.bname join
dep d on a.ano = d.ano
where city = 'Jaipur';
```

Q.

emp(ename, street, city)  
works(ename, cname, sal)  
company(cname, city)  
manager(ename, mname)

(a) Find names of all employees who works for first bank corporation.

Ans -

```
select ename from works
where cname = 'First bank corporation';
```

(b) Find average salary of each company.

Ans -

```
select avg(sal), cname from works
group by cname;
```

(c) Find all emps who earned b/w 5000 and 10000 in descending order.

Ans -

```
select ename, sal
from emp works
where sal between 5000 and and 10000
order by desc;
```



(d) Display ename & then mname.

Ans -

```
select ename, mname
from manager;
select * from emp;
```

(e) Find name of all employees who do not work for small bank.

Ans -

```
select ename
from works
where cname <> 'small bank';
```

(f) Find name & cities of all employees who work for first bank.

Ans -

```
select e.ename, city
from emp e join company works w
on e.city = w.city e.ename = w.ename
where cname = 'first bank';
```

or

```
select ename, city
from emp
where ename in (select ename
from works
where cname = 'First bank'));
```

(g) Display ename & city of all employees which earn average salary of company first bank.

Ans -

```
select ename, city
from emp
where ename
```



## ★ Embedded (static) and Dynamic SQL ⇒

Even though SQL is a powerful declarative language. Its expressive power is less than a general purpose programming language.

Eg - Printing a report, taking input from GUI can not be done in SQL. Therefore SQL can be used with a programming language to provide a program more control. The language in which SQL can be attached to is known as HOST language (C++, Java, python etc.) and the elements, commands of SQL that can be written in host language is called Embedded SQL.

A special preprocessor is used to convert SQL statements to procedure and function calls that are understood by the host language compiler. This compiler identifies SQL in a programming language using EXEC SQL.

Syntax:-

```
EXEC SQL
```

```
<sql statements>
```

```
END EXEC
```

The preprocessor provides the SQL statements to the data base system which returns results one tuple (row) at a time.



For a relational Query these tuple must be opened & fetched by declaring a cursor.

eg-

EXEC SQL

declare cursor c1 for  
select first name, last name  
from employees;

END EXEC

char fn[20];

char ln[20];

EXEC

SQL

open c1

END

EXEC

EXEC

SQL

fetch c1 : fn, : ln

END

EXEC

eg -

EXEC

SQL

insert

into

employees

(first name, lastname)

values

('ABC', 'XYZ');

END

EXEC

★

Dynamic SQL :-

allows

at

to

run

This

component of SQL

allows to formulate, produce queries

at run time.

produce

queries



```

void main ()
{
    EXEC SQL BEGIN DECLARE SECTION
        char fname[16], lname[16];
        float sal;
    EXEC SQL END DECLARE SECTION
    printf("enter salary");
    scanf("%f", &sal);
    EXEC SQL DECLARE cn CURSOR FOR
        SELECT first name, lastname
        FROM employees
        WHERE salary > :sal;
    END EXEC
    EXEC SQL OPEN cn
    EXEC SQL FETCH cn INTO :fname, :lname
    WHILE (SQLCODE == 0)
    {
        printf("%s %s", fname, lname);
    }
}

```

Dynamic SQL: —

```

EXEC SQL BEGIN DECLARE SECTION
    char sqlstr[200];
EXEC SQL END DECLARE SECTION
EXEC SQL PREPARE query FROM :sqlstr
EXEC SQL EXECUTE query

```



## Static SQL

- (i) SQL is processed at compile time.
- (ii) Faster and efficient
- (iii) No flexible
- (iv) Has error checking

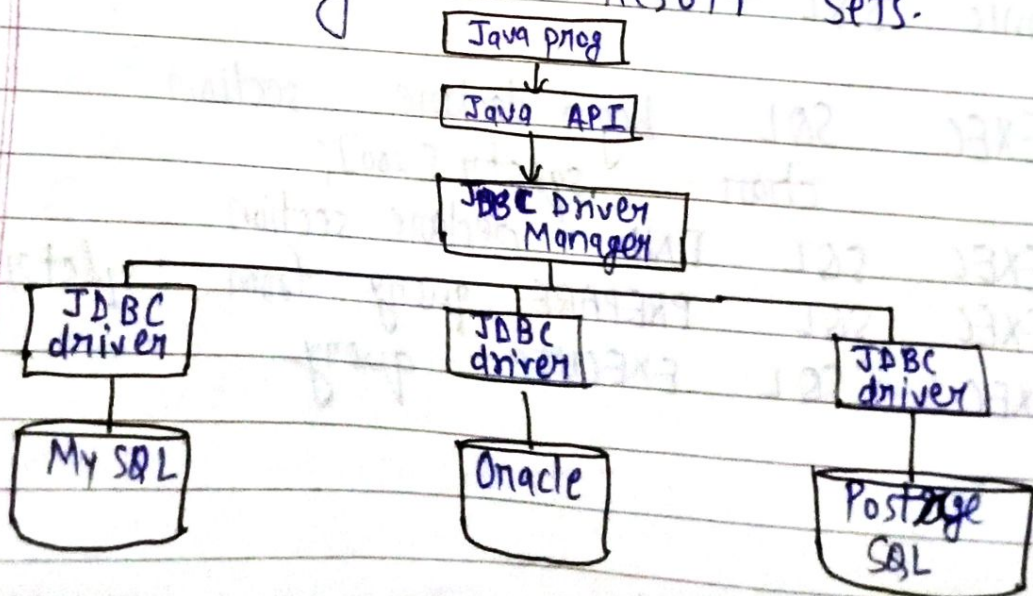
## Dynamic SQL

- SQL is processed at run time.
- Slow and inefficient.
- Flexible
- No error checking.

### JDBC :— [Java Data Base Connectivity]

It is a standard API (Application programming interface) which provides a set of classes containing function call to connect a java program to a database server. It has the following major step:—

- (i) Making a connection
- (ii) creating SQL statements
- (iii) Executing SQL statements
- (iv) Retrieving Result sets.





JAVA API  $\Rightarrow$  It provides an intermediate for the program and the driver manager.

Driver Manager  $\Rightarrow$  It is an intermediate which provides an access to JDBC driver to Java API.

### STEPS TO CONNECT :—

- (1) Import necessary Packages  $\Rightarrow$  This steps require to provide necessary packages containing ~~var~~ various classes which allows to connect to the data base.
- (2) Register the JDBC Driver  $\Rightarrow$  This steps initialises the driver to ~~the~~ begin communication channel to the database.  
 JDBC Supports 4 type of drivers
 

$\rightarrow$ Type 1	Jdbc - odbc
$\rightarrow$ Type 2	Native
$\rightarrow$ Type 3	protocol
$\rightarrow$ Type 4	thin
- (3) Open connection  $\Rightarrow$  Create a connection object to represent physical connection to database.
- (4) Build statements  $\Rightarrow$  Use the statement interface to generate and build SQL statements.



(5) Fetch ResultSet  $\Rightarrow$  Resultset contains various methods to retrieve rows one by one.

**Active Database:** — In this database which can respond to internal and external events of the system. This active behavior is defined in terms of ECA Rules which are formulated during database creation. Generally, active dbs are implemented through stored programs known as triggers. Active db can have a built-in architecture where it is integrated into the database or can have a layer architecture where it exists above the db.

★ **Database Trigger:** — It is a stored procedure which gets executed automatically when a certain event occurs in the database. It is basically a side-effect of the modification to the db. To create a trigger two requirements must be met which are given by ECA rule [Event Condition Action].

- (a) Event which must occur for trigger to execute and condition that must be satisfied.
- (b) Action which will be performed after the trigger execute



Triggers in SQL are two type

- (i) Row level
- (ii) Table level

- (i) Row level:- This trigger executes for each row.
- (ii) Table level:- This executes once.

### Syntax

```
(create [on replace] Trigger tname
[ Before | After | instead of ]
[ insert | update | delete ] [ of column ] on object
[ Referencing new row as n oldrow as o ]
[ for each row ]
when [ condition ]
Declare variable section
Begin
    body of trigger.
End
```

Q. Create a trigger which calculates total salary from commission %.

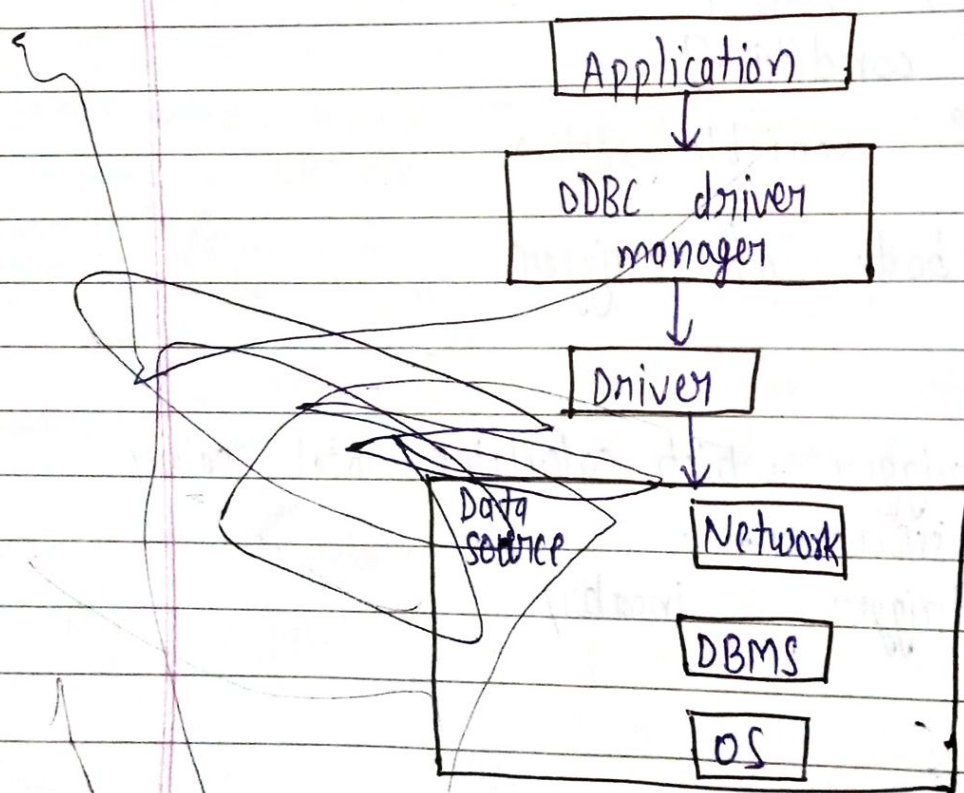
Ans-  
create trigger insalary  
before  
insert  
for each row  
Begin  
total salary = sal \* (1 + comm\_pct)  
end



Q. Create a trigger which stores raises given to each employee

A- create trigger raises  
 before after  
 insert update  
 Referencing new row as n old row as o  
 begin for each row  
 declare n int  
 begin  
 $n = n.salary - o.salary$   
 end

ODBC [Open data base connectivity] :-



(i) Application:- It is the client application which request call to ODBC.

(ii) ODBC driver manager:- It manages the connection for the application and loads the drivers to initiate the connectivity



(iii) Driver:— It is a library which responds to the connectivity request and processes ODBC function calls. It also translates SQL into standard native calls.

(iv) Data source:— It is a combination of operating system, network protocol & DBMS.

## ★ Views ⇒

It is a Query stored under an associative name. It is a form of virtual table for organising information.

Syntax:—

```
create view vname as
select
from
where
```

Eg- create view v as  
select ename en, dname dn  
from emp e join dept d on e.did = d.did;  
select \*  
from v;

## Uses/Properties of View ⇒

- (i) It is a stored Query which does not take up as much space as a table.
- (ii) It summarises data from multiple relations.
- (iii) It is used to organise & simplify complex queries.



(iv) It restricts data access.

### Views in DataBase Security $\Rightarrow$

- (i) It provides better security by allowing only certain columns from tables to be visible.
- (ii) Difficult to update, insert values.
- (iii) Restricts user access.



## Schema Refinement & Normal Form

Sem	Subject	dept	Faculty	Qualification
1	phy	CS	ABC	En
2	Math	EY	XYZ	En
3	Op	CS	MNO	En
4	DBMS	EE	MNO	En
4	DN	CS	ITK	En
3	Math-3	CE	XYZ	T.C.

### \* ANOMALIES $\Rightarrow$

These are the problems which occur due to a poorly design database with unnormalised table. It has 3 types.

#### (i) Insert anomalies:—

It is the inability to add new data without providing complete set of information i.e. new information can only be inserted if entire information is provided.

#### (ii) Update anomalies:—

This occurs due to data inconsistency and partial update.

#### (iii) Delete anomalies:—

This anomalies occur when deletion of information result in lose of other information as well.



## \* Functional dependency $\Rightarrow$

It is a type of constraint which generalises the notion of key.

Consider attributes  $\alpha, \beta$  in a relational schema  $R$ .  
Functional dependency  $\alpha \rightarrow \beta$  is true if for a relation  $r$  in  $R$ ,  $\alpha \rightarrow r$ ,  $\beta \rightarrow r$  and if all the tuples  $t$

$t_1[\alpha] = t_2[\alpha]$  then the following must be true.  $t_1[\beta] = t_2[\beta]$

For example

	Rollno.	studentname
$t_1$ :	7	Anil
	8	Babu
$t_2$ :	7	Anil

## Closure of functional dependency :-

Given a set of functional dependency  $F$  closure ( $F^+$ ) are all the dependencies that can be logically implied by  $F$ .

A general set of inferences are given by Armstrong Axioms

## Armstrong Axioms $\Rightarrow$

These are rules of inferences to obtain closure of functional dependency  $F$ .

- (i) Reflexivity :-  $\alpha, \beta$  are attribute  $\beta \subseteq \alpha : \alpha \rightarrow \beta$
- (ii) Augmentation :- if  $\alpha \rightarrow \beta$ ,  $\gamma$  is a attribute,  $\gamma \alpha \rightarrow \gamma \beta$
- (iii) Transitivity :- if  $\alpha \rightarrow \beta$ ,  $\beta \rightarrow \gamma$  ;  $\alpha \rightarrow \gamma$



(iv) Union:— if  $\alpha \rightarrow \beta$ ;  $\alpha \rightarrow \gamma$  :  $\alpha \rightarrow \beta\gamma$

(v) Decomposition:—  $\alpha \rightarrow \beta\gamma$  :  $\alpha \rightarrow \beta$ ,  $\alpha \rightarrow \gamma$

(vi) Pseudotransitivity:—

$\alpha \rightarrow \beta$ ,  $\beta\gamma \rightarrow \gamma$  :  $\alpha\gamma \rightarrow \gamma$

Q.  $R = \{A, B, C, G, H, I\}$

given FDs

FD1:  $A \rightarrow B$

FD2:  $A \rightarrow C$

FD3:  $CG \rightarrow H$

FD4:  $CG \rightarrow I$

FD5:  $B \rightarrow H$

$F^+$  {  $A \rightarrow B$ ,  $B \rightarrow H$  :  $A \rightarrow H$   
 $CG \rightarrow H$ ,  $CG \rightarrow I$  :  $CG \rightarrow HI$   
 $A \rightarrow B$ ,  $A \rightarrow C$  :  $A \rightarrow BC$   
 $A \rightarrow C$ ,  $AC \rightarrow CG$  :  $AC \rightarrow I$

Q. Emp\_project (emp-id, project-id, howsgiven, projname, emname, projlocation)

dy —

FD1: emp-id  $\rightarrow$  emname

FD2: project-id  $\rightarrow$  projectname

FD3: project-id  $\rightarrow$  projlocation

FD4: emp-id, project-id  $\rightarrow$  howsgiven

FD5: project-id  $\rightarrow$  projectname, projlocation

Types of Functional dependency:—

① Trivial FD  $\Rightarrow$  A functional dependency  $\alpha \rightarrow \beta$  is trivial if  $\beta \subseteq \alpha$ .  
eg pid emp-id  $\rightarrow$  emp-id pid



② FULL FD  $\Rightarrow$  If there exists dependency  $\alpha \rightarrow \gamma$ , there is an attribute  $A$  in  $\alpha$  such that  $\{\alpha\} - A \rightarrow \gamma$  is false. then  $\alpha \rightarrow \gamma$  is full FD.

eg- empid, projectid  $\rightarrow$  hoursgiven  $\checkmark$   
empid  $\rightarrow$  hoursgiven  $\times$   
on projectid  $\rightarrow$  hoursgiven  $\times$

③ Partial FD  $\Rightarrow$  If there exists dependency  $\alpha \rightarrow \gamma$ , there is an attribute  $A$  in  $\alpha$  such that  $\{\alpha\} - A \rightarrow \gamma$  is True then  $\alpha \rightarrow \gamma$  is partial FD.

eg- projectid, empid  $\rightarrow$  ename  $\checkmark$   
projectid  $\rightarrow$  ename  $\times$   
on empid  $\rightarrow$  ename  $\checkmark$

④ Transitive FD  $\Rightarrow$  If  $\alpha \rightarrow \gamma$  and  $\gamma \rightarrow \beta$  then  $\alpha \rightarrow \beta$  is transitive dependency.

⑤ Multivalued FD  $\Rightarrow$  For a relation  $R(\alpha, \beta, \gamma)$ , if  $\alpha \twoheadrightarrow \beta$ ,  $\alpha \twoheadrightarrow \gamma$  that is  $\beta, \gamma$  are dependent on  $\alpha$ . ~~and~~ Such that  $\beta, \gamma$  are independent of each other then such dependencies are multivalued.

## ★ NORMALIZATION $\Rightarrow$

It is a database schema design / refinement process which restructures tables in a data base. It organises the tables to reduce redundancy and ambiguous dependencies.



among the tables. It is given by and implemented through normal forms:-

(i) ~~1st~~ 1-NF :-

A table is in 1-NF if all the values of attribute in a table are atomic (individual)

Q. dept ( dname, dno, dmgr-id, dloc )

Research	5	12	delhi, mumbai
IT	4	13	Bangalore, pune
HQ	1	11	Jaipur
Market	2	14	Chennai

Q - The above table is not in 1-NF.

Research	5	12	delhi
IT	4	13	Bangalore
HQ	1	11	Jaipur
Market	2	14	Chennai
Research	5	12	Mumbai
IT	4	13	Pune

Now above table is in 1-NF.

(ii) 2-NF :- A relation is in 2-NF if it is in 1-NF and every non-prime attribute is fully functionally dependent on prime attribute.

prime attribute  $\Rightarrow$  If an attribute is a member of candidate key then it is a prime attribute otherwise it is



non-prime attribute.

Decomposition  $\Rightarrow$  It is the process of dividing a relation into sub-relations. It has the following properties:—

- (i) Attribute Preservation  $\Rightarrow$  Decomposition must happen in a way so that there is no loss of attributes.
- (ii) Dependency Preservation  $\Rightarrow$  If a relation are broken down into  $R_i$  (sub relation) has  $N$  dependency then  $R_i$  must abide to at least one dependency and collectively  $R_i$  must follow all the dependencies.
- (iii) Non-additive Dependency  $\Rightarrow$  Union of sub relations returns the original relation.

Types of Decomposition:—

- (i) Lossy
- (ii) Loseless

- (i) Lossy :— If decomposition results in loss of data or extra information then it is lossy.
- (ii) Loseless :— For a relation  $R$  decompose to  $R_1, R_2, \dots, R_n$  then If  $R_1 \bowtie R_2 \bowtie R_3 \dots \bowtie R_n$  is  $R$ . Then it is loseless.



## Normalization :-

It is the process of organizing the data in the database.  
It is used to minimize redundancy from a relation or set of relation.

It is also used to eliminate Anomalies (Insertion, update, delete).  
It is a decomposition of table & done by basic of functional dependencies.

1NF :- If a relation contains an atomic value.

2NF :- If it is in 1NF & all non-key attributes are fully functional dependent on the primary key.

3NF :- If it is in 2NF and no transitive dependency exists.

$\alpha \rightarrow \beta$   
either  $\alpha$  is superkey or  $\beta$  is prime attribute.

BCNF :- If it's in 3NF and  $\alpha$  should be superkey irrespective  $\beta$ .

4NF :- If it's in Boyce Codd NF & has no multi-valued dependency.

5NF :- If it's in 4NF and not contains any join dependency & joining should be lossless.



## SQL

→ It is a relational database management system.

→ These databases have fixed or static or predefined schema.

→ Not suited for hierarchical data storage.

→ Best suited for complex queries.

→ Vertically scalable

→ follows ACID property

→ It is table Based structure.

→ Eg- MySQL, Oracle etc.

## NoSQL

Non-relational or distributed database system.

They have dynamic Schema.

Best suited for hierarchical data storage.

Not so good for

Horizontally Scalable

Follows CAP

[Consistency, availability, partition tolerance]

It is a key-value pairs, document-based, graph databases.

Eg - MongoDB, BigTable, HBase,