14.06.2017

MODULE-I

INTRODUCTION TO DBMS -

Overview -

- **Data** - It is collection of facts . eg. word files, image, audio files etc.
- **DataBase** - It stores data in an organized manner so that it can be easily accessed manipulated and retrive when it is necessary.
  - symbolic representation of DataBase

- **DataBaseManagement System(DBMS)** -
  It is software which is responsible for creating and manging DataBase.

→ **Applications of DataBase**

- Banking applications
- ecommerce
- Reservation systems
- Web Applications
- Finance Applications
- Manufacturing industries
- Hospitality Applications
- Universities
- Military Applications
- Central library Applications.

20.06.2017

⇒ List of DataBase (DB) -

1) Oracle 12C (product of oracle C → cloud.) These are used by cloud application.
- This DB is meant to work in cloud environment.
- Oracle 12C stores data in RelationaldataBase Management System (RDBMS). (tables)

2) Microsoft SQL server (product of Microsoft) - This stores data in RDBMS. (It runs only on server)

3) MySQL - (product of sun microsystems - This stores data in RDBMS. (It can Run of device, server, system etc)

*4) MySQL Enterprise Transperant Data Encryption - RDBMS (Security)

AES - Advanced Encryption Standard Algorithm - occupies 256 bit
↳ This have capacity to Encrypt as well as decrypt data

- This DBMS is responsible for encrypting data at rest using AES (Advanced Encryption standard Algorithm) having key length of 256 bits.

5) IBM DB2 - (product of IBM) - To store RDBMS

6) SAP sybase ASE (mobile application) - (product of oracle) - stores data in RDBMS.
SAP - Systems Applications products
ASE - Adaptive server Enterprise
- Most famous Enterprise application that is used currently.

7) PostgreSQL (or) postgres - stores data in ORDBMS (Object Relational DataBase Management System).
(This is used in web applications as Yahoo, Skype etc.
NOSQL is used in FB or other social Networks).

8) Maria DB - "open source" - RDBMS.
- It will have GPL (General public Liscence)
- Special feature is "Query Optimisation"

9) Yera Data - This is Datawaire House.
- stores data in RDBMS

21.06.2017

⇒ File System Vs DataBase Management System -

(i) Data Redundancy-

Duplicate data can be present at different location in file system where as in DataBase it is kept to minimum.

(ii) Data inconsistancy-

- In file system the modifications are made at a particular location then it will not be reflected where similar data is store such data is called as Inconsistent data.
- But, incase of database, data modification made at a paticular location will automatically update its corresponding data items also. As a result of this, inconsistent data can be avoided.

(iii) Integrity Constraints :-(Rules)

In order to maintain integrity of data, we can impose certain rules on the individual tables or on the entire DataBase Whereas, these rules cannot be imposed in filesystem.

(iv) Concurrent Access of Data -

- Using Database multiple users can access the same data item at a particular point of time.
- This concurrent access is not possible in the filesystem.

(v) Data Security -

- To maintain data security database Administrator will provide different levels of access rights to different users based on the requirement.

(vi) Atomicity -

- Even after any device fails and if the data is available then it is called automicity.
- In the case of Database, periodic backup of data will be maintained in order to assure atomicity of data.

(vii) Difficulty in Accessing Data -

- In the case of file system in order to retrive any data item we need to traverse through a length path or write a complex program.
- In the case of Database, we can easily retrive the data by using simple queries.

⇒ Advantages of Database -

1) Reductancy is minimized.
2) Data Inconsistancy is avoided.
3) Constraints can be imposed on the entire database (or) any part of the database.
4) Concurrent access of data is provided.
5) Data Security is also provided.
6) Data ~~Atto~~ Atomicity is maintained.
7) Data can be easily accessed from the database.

22.06.2017

⇒ Storage of Data -

1) Relational Model
2) Data Abstraction
3) Data Independence

→ Relational Model -

- It is the data model which stores data in the form of relations (tables).
- Every table should contain unique name called as ~~Entity~~ name or name of the entity.
- Depending on the number of attributes to be added into the table, no of columns will be decided.
- Each row within the table is called as a record.
- The overall the structure of the table is called as schema.
- The data ~~inse~~ present in a table at a particular point of time is called as instance.

SNo.	RollNo	name	Section	Branch	year
1	1	xx	BH	CSE	III
2	2	yy	BH	CSE	II

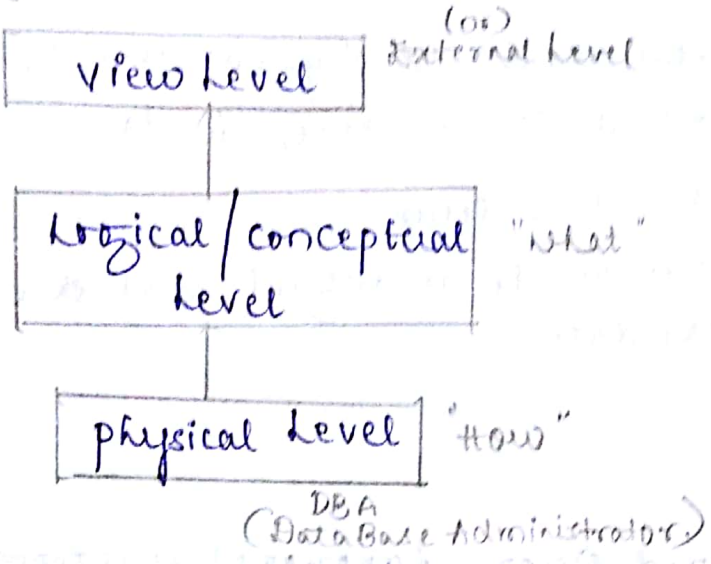
Record

column/attribute

schema/structure

→ Data Abstraction -

- Data abstraction / view levels of Data



- Physical Level -
  - This level deals with how to store data within database.
  - Very low level of abstraction is provided at this level.
  - It is duty of DBA (database administrator) to decide data models and the structure of database at this level.
- Logical or conceptual Level -
  - This level provides little more abstraction when compared with physical level.
  - It is duty of application programmers to decide what data is to be stored on the database.
  - Even the suitable query language to manage database will be selected at this level.
- View Level or External level -
  - High degree of data abstraction is provided at this level
  - It is place where end users actually view the database.
  - Depending on the queries send by end users only a certain portion of database will be visible to the enduser.

→ Data Independence-

23.06.2017

- Data Independence is provided at two levels.

1) physical data Independence -

- Any modifications made at physical level of database should not effect logical or conceptual level.

2) Logical data Independence -

- Any modification made at logical level should not effect view level of the database.

lab

Queries-

SQL (Structured Query Language) is categorised into following types -

1) Data Definition Language (DDL)
2) Data Manipulation Language (DML)
3) Data Control Language (DCL)
4) Data Query Language (DQL)
5) Transaction Control Statements (TCS).

1) Data Definition Language (DDL)-

- DDL commands are responsible for managing the schema (structure of Table)

- Create -

- This command is used to create a new table with unique table name.

- Syntax -

```
create table <table-name> (column1 type, col2 type...);
```

- Eg -

Data Types -

- number(size) number(10) → 10 digits will be allocated
  This data type can store only integer values
- number(size, precision) number(10, 3)
  number can be 10 digits and precision will be after 3 digits.
  eg:- 100.000

- char(size)
  This can store character data.
- Varchar(size) (This will accept characters as well as numbers)
- date (This data type is used to accept only date).

Eg of DDL create -

Create table 2015CSEB4(rollno number(10), name char(25), address varchar(100), Doj date);

2015CSEB4

Roll_no	name	Address	joining date

- Here, after creating table, if there are no errors in the command, then ~~we~~ it will be displayed that the table is created.

- **Drop -**

This command is used to delete or drop entire table (ie) schema or structure of table.

Syntax -

```
drop table <table name> ;
```

eg -

drop table 2015CSEB4

- **alter -**

This command can add a new column, delete an existing column (or) change the data type of a paticular column

✓ syntax for adding a column -

```
alter table <table name> add column <column name> <type>;
```

eg -

alter table 2015CSEB4 add ~~sno~~ column sno number(10);

✓ Syntax for delete/drop column -

```
alter table <table name> ~~delete~~ drop column <column name>;
```

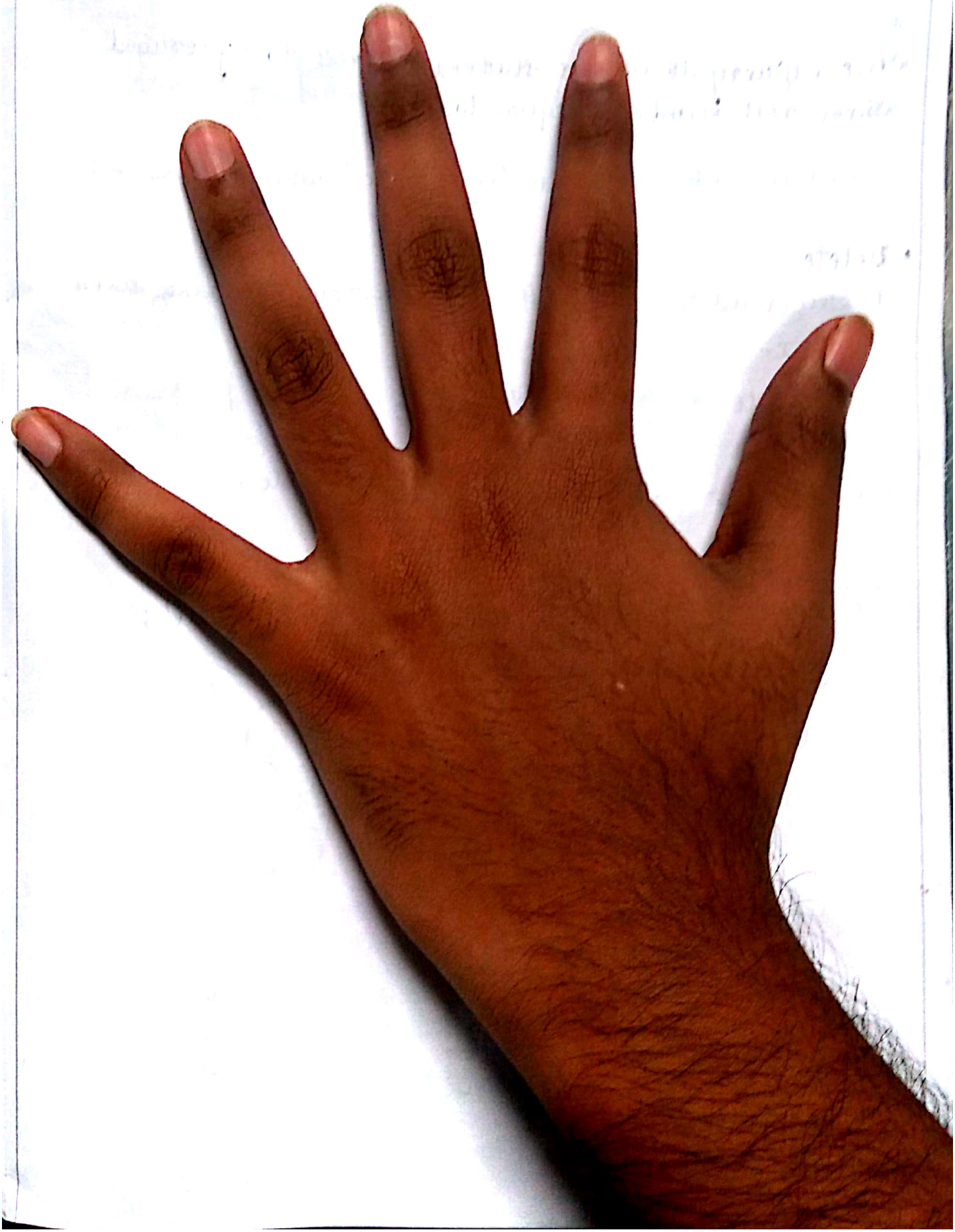
eg - alter table 2015CSEB4 drop column sno;

✓ syntax for changing data type -

```
alter table <table name> modify (<column name> <new_type>);
```

eg - alter table 2015CSEB4 modify (rollno varchar(10));

2) Data Manipulation Language



- Update -

To update values in the table

update <tablename> set <col=value> where '<col=values';

eg-

Write a Query to update student address to Hyderabad where roll number is equal to 1.

update CSEBy set add='Hyderabad' where rollno=1;

- Delete -

(i) Delete all the records in a table without altering schema

Syntax-

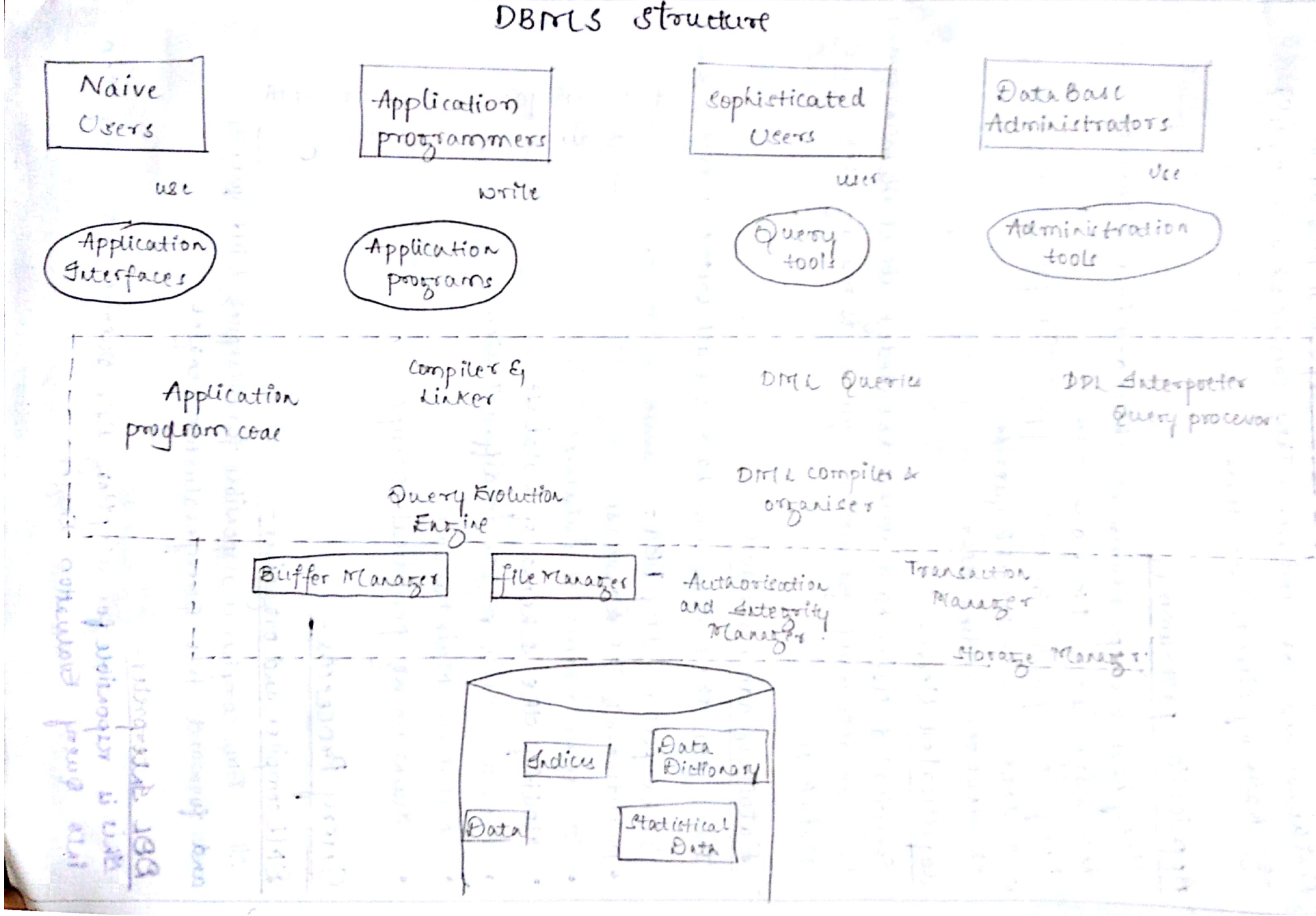
delete * from <table name>;
(* → all)

(ii) To delete a paticular record from a table based on given condition.

Syntax-

delete from <tablename> where col='value';

DBMS structure



- Naive Users or Novice Users-

- These have no knowledge about application program or Database and are end users, they can access users by GUI
eg- Railway clerks, Bank clerks.

- Application programmers -

- They have good programming skills and have moderate knowledge about DB managemers.
They are responsible for writing application programs that work together with the DB queries
eg- developer, coding team

- Sofisticated Users -

- Sofisticated users have sound knowledge about DB managemen
They are responsible for writing queries to manage the Data Base.

- DataBase Administrators.

- These are the persons who have overall control over DataBase.

Tasks performed by DBA:
- partitioning the Database
- To define schema of Database
- Modify the schema of Database
- Provide access rights to different users
- Database Maintanance.
- Maintaining periodical backups

Query Processor -

DML compiler and organiser-

This DML compiler is responsible for accepting DML queries and forward into Query evaluation engine

DDL Interpreter-

This is responsible for acepting DDL Queries and sending them into Query Evaluation Engine.

Query Evaluation Engine -

This Query Evaluation Engine converts machine understandable form and it is also responsible for managing storage activities performed over the Database

→ Storage Manager -

Buffer Manager -

- Buffer Manager is responsible for fetching Data from the dataBase and send it to main memory of a computer system.
- It is the duty of buffer manage to handle a large amounts of data and retrive the data from DB.

File Manager

Responsible for storing Data in DB and manage space over the Data base

Authorisation and Integrity Manager -

- This part is presponsible for checking/verifying user details before they access the database
- It is responsibility of integrity manager to make sure that the data stored inside DB is consistant and not modified

Transaction Manager -

A transaction manager is responsible for controling various transactions that occur over the database.

Disk storage/DataBase -

Data -

- Actual information that is stored in the database
  eg - Audio, video, multimedia, textual data, word doccuments

Indices/index -

It shows the location where the actual data is stored.

Statistical Data - Statistical terms related to each data item is called as statistical data.
eg - no of views, no of likes, no of downloads.

Data Dictionary - This can store meta data.
meta data is data about data. It maintains information about schema of the database and further details.

TRANSACTION MANAGEMENT -

- It is a program that can access a data item or modify the exisiting item. is called transaction.

The duty of transaction manager is to check that each transaction follows four properties.

- Atomicity
- Consistancy
- Isolation
- Durability

→ **Atomicity -**

In any transaction either ~~the~~ all the lines should be reflected in the database or none of them should be reflected

→ **Consistancy -**

The sum of values before the transaction should be equal to the values even after the transaction.

→ **Isolation -**

Every transaction should be performed in an isolated manner Any transaction should not be visible to other transaction carried out on a data item

→ **Durability -**

The data that is stored in the database should be available for long period of time.

Even after Database crash also the data should be still available.

29.08.2019

Data Modeling and Data Models-

Data Modeling-

It is the process of representing system design in the form of diagrams/pictures using symbols that are easily understandable

Importance of modeling-

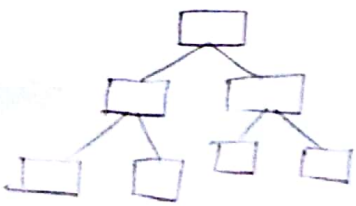
- To make the design to be easily understandable to end users, designers and application programmers.
- If the modeling is done properly with minimal problems implementation part will be easier.
- By seeing the data models different stake holders will have idea about what is to be actually added to the database.

Data Models Evolution

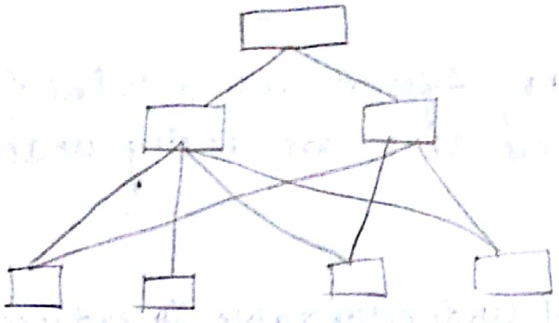
(i) File system
(ii) Hierarchical Model - Tree
(iii) Network Model - Graph
(iv) Relational Model
(v) Object oriented Model
(vi) Object Relational Model.

(ii) Hierarchial Model -

- In Hierarchial Data model Data will be stored in the form of tree structure.
- A parent can have any no of child data items, but a child can have only one parent
- In order to find a paticular data item, the entire tree structure should be traversed beginning from the root node.
- ~~With~~ If any modifications are to be made, the entire tree structured will be locked.

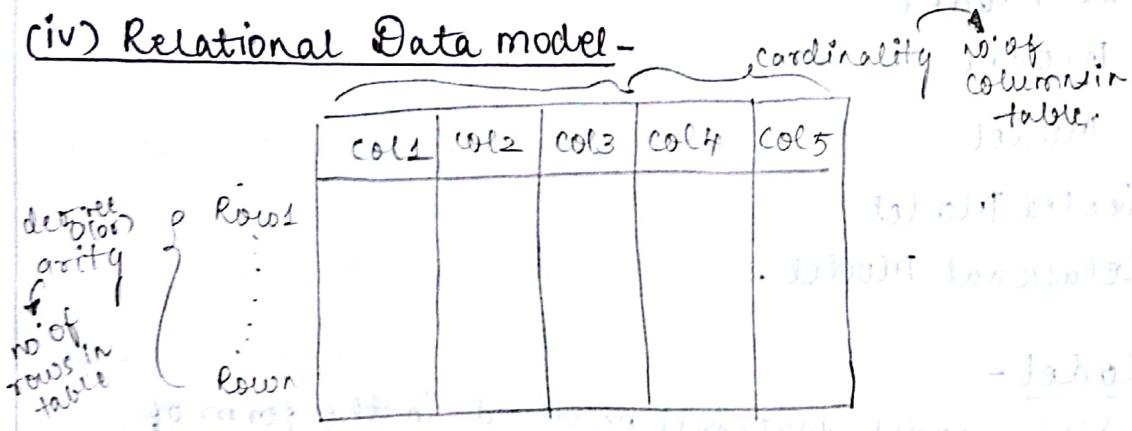


(iii) Network Data Model -



- Network model data will be stored in the form of graph structure.
- A parent can have any number of child data items and every child can have any number of parent data items.
- Remaining are same last 2 points from ~~the~~ Hierarchial model.

(iv) Relational Data model -



- In Relational Data model, data is stored in the form of table
- RDBMS can store the data other than images, animations, graphics etc.
- Cardinality of a table = no of columns
  arity of a table = no of rows in table.
  or degree.

v) Object oriented Data Model -

- This data model is suitable for storing high end data items like images, graphics, videos etc.
- To manage data within the database an Application program interface in sufficient we need have any spl Query language.
- In this data model similar data items can be linked and can be stored permanantly.

vi) Object Relational Data Model -

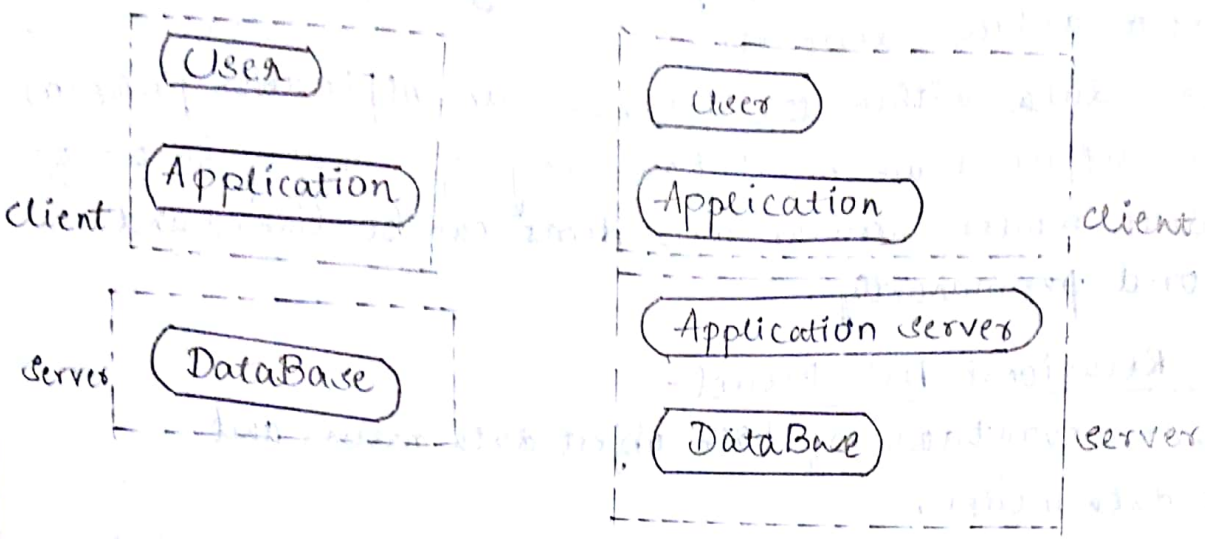
- It combines advantages of both object data model and relational data model.
- It stores data in the form of tables similar to relational data model
- We can create our own datatypes to customize API (Application program Interface) with the database.
- Both Query Language and API will be used to manage the DataBase.

(vii) ER-Model -

- Entity Relationship model
- This data model explains about how data is to be organised at design level.
- We use various symbolic representation to represent the data that is to be stored in the database.

04·07·2017

Two Tier, Three Tier Architecture -



Two tier Architecktere -

- Here, client can directly interact with Database by sending request for local server.

Three-tier Architecture -

client cannot directly interact with the resources on Database present at some remote location.

- Request sent by the client will be received by application server.
- Internet - Internet connection is to be present in order to perform client and server interaction

Data Design -

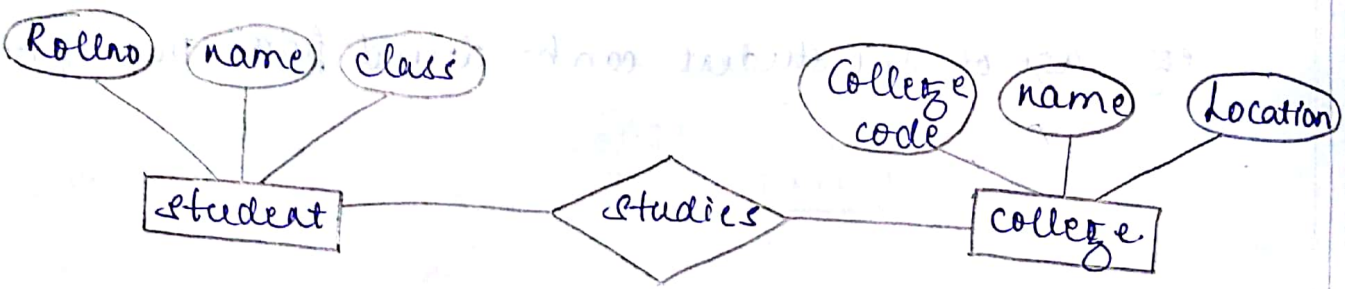
(i) Conceptual Level - ER-Diagrams
(ii) logical level - Data models will be selected, How to organize data is decided.
(iii) physical level - schema is created.
Data is entered.

ER- Diagrams-

(Entity Relationship diagram)

- Entity- Any object that exists in the real world is called as entity.
  - Entity is represented as '▭'.
- Attribute- It determines properties of an entity, represented as '⬭'
- Relation- It provides association between two or more entities.
  - It is represented by '◇'.

Eg-



TYPES OF ATTRIBUTES-

1) simple vs complex attributes

- An attribute that can be subdivided into attributed called as ~~sub~~ ~~simple~~ complex attributes.

  Eg- name

  name can be further divided into
  - first name
  - last name .. etc.

- An attribute which cannot be further divided, ~~into~~ is called simple attributes.

  Eg- phone number

(2) Single and Multivalued Attribute

- Single valued attribute - Attribute which can hold only a single or one value

  eg- Roll no of a student.

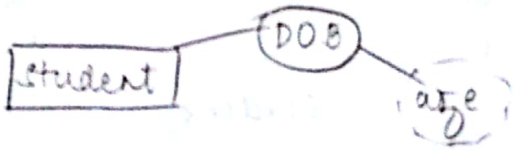
- Multivalued attribute - An attribute that can hold more than one value then it is multivalued / composite valued

  eg- phone number.

3) Derived Attribute -

- If one attribute further can be derived from some other attribute is called Derived attribute.

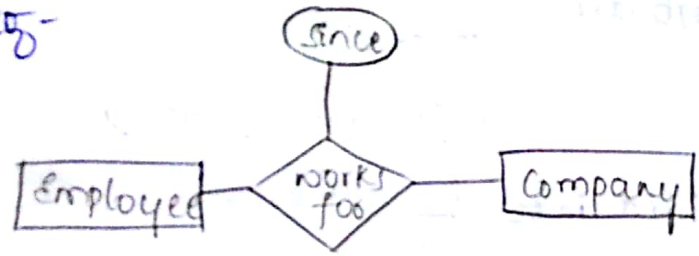
  eg- age of the student can be derived from Date of Birth



4) Descriptive attribute -

- An attribute added to a relation is called as descriptive attribute

eg-



since is descriptive attribute.

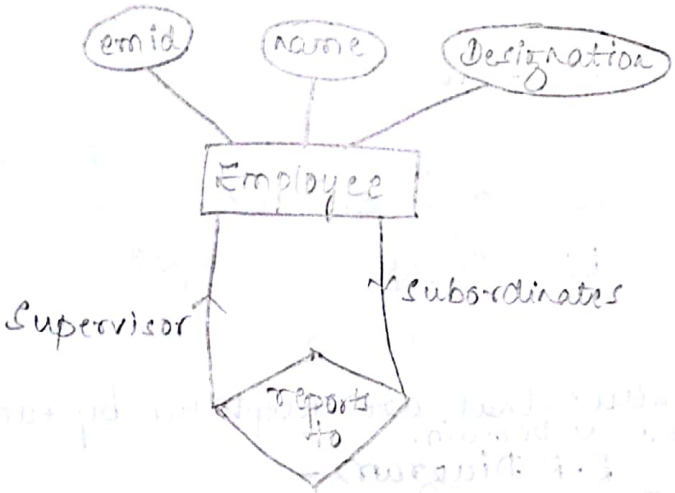
ER-Diagrams -

Types of Relation -

Unary -

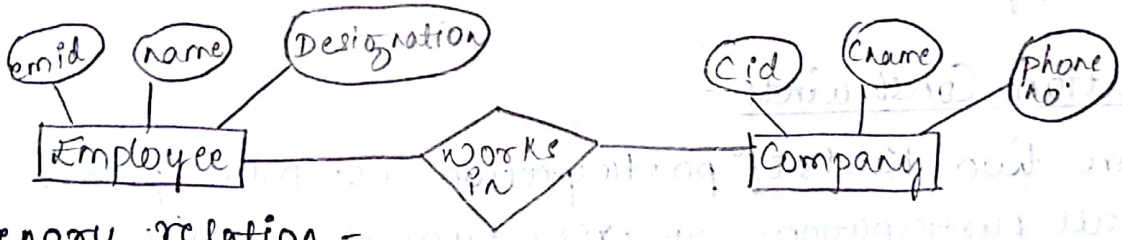
- Some times it is necessary to connect entity belonging to the same relation set. Then we use unary relation to connect them.
- Along with this the roles played by the entity should also be mentioned. This is called as role indicator.

Eg -



Binary relation -

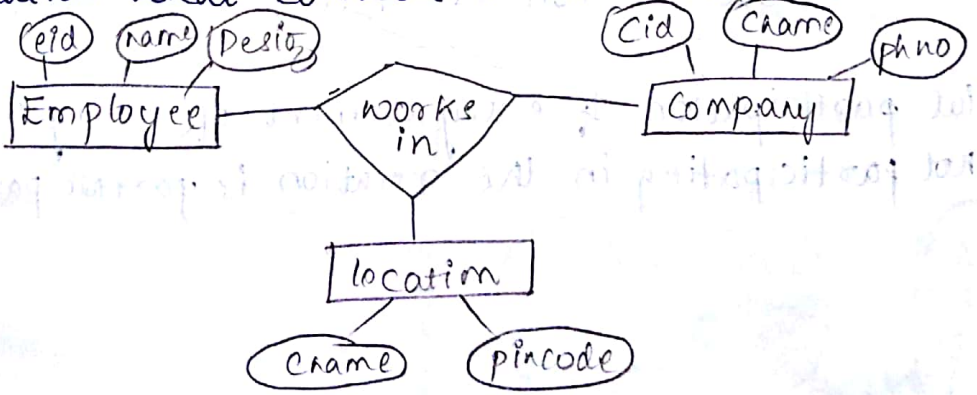
- A relation that connects to entities



Terenary relation -

- A relation that connects three or more entities.

Eg -



Entity set -

Collection of similar entities is called as Entity set.

We represent Entity set as "E" = {$e_1, e_2 \ldots e_n$}

(E: entity set; $e_1, e_2 \ldots e_n$: entities)

Relation set -

Two relations are said to be similar if they connect similar entities.

Then, collection of similar relations is called as relation set

Domain - Set of possible values that are acceptable by each attribute is called as Domain.

* Additional features of E-R Diagram -

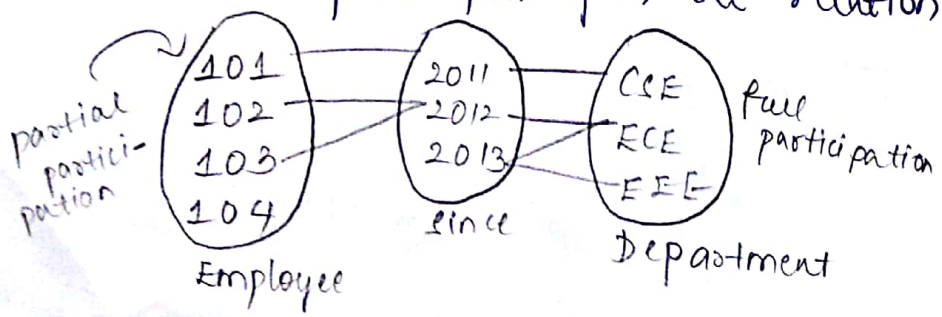
- Participation Constraints
- Key Constraints
- Mapping Constraints.

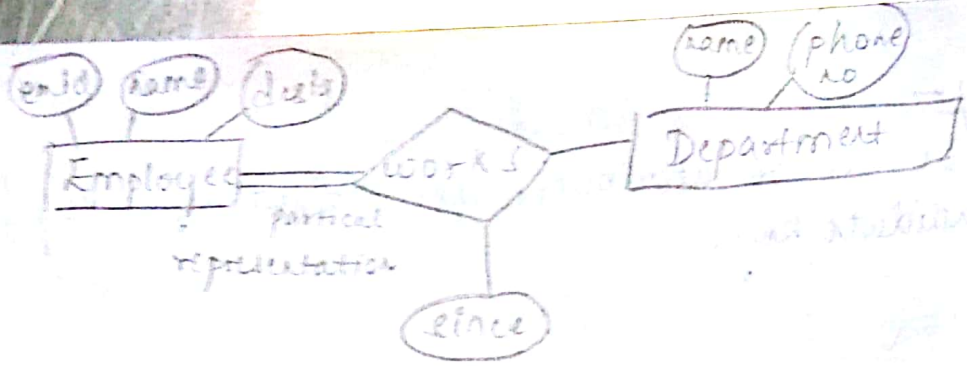
Participation Constraints -

There are two kinds of participation constraints present

(i) Full participation - If every element of entity is participating in the relation is Full participation

(ii) Partial participation - If every element of entity set is not participating in the relation is partial participation





10.07.2017

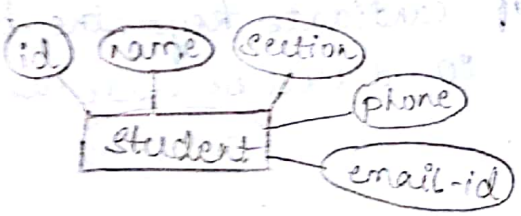
Key Constraints-

It is an attribute or set of attributes that can uniquely identify a record from the table.

Super key-

Set of all possible keys of a particular table or entity is called as super key.

eg-



I: {id} {name} {section} {phone} {email-id}

II: {id, name} {id, section} {id, phone} {id, email-id}
{name, section} {name, phone} {name, email-id}
{section, phone} {section, email id}
{phone, email-id}

III: {id, name, section}
{id name, phone}
{id name, email}
{name, section, phone}
{name, section email}
{section, phone, email}

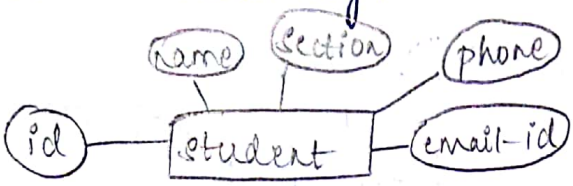
IV: {id, name, section, phone}
{id, name, section, email-id}
{name, section, phone, email-id}

V: {id, name, section, phone, email-id}

Candidate key -

It is the minimal set of attributes derived from super key is called as candidate key.

eg -



{id} {phone no} {email-id} are the candidate keys of table.

Primary key -

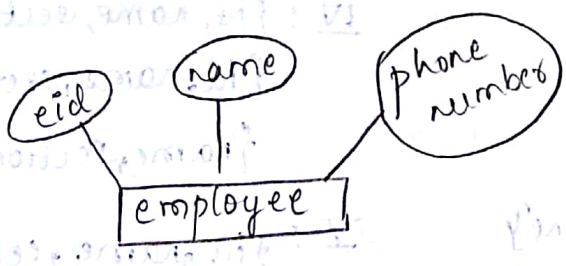
- It is one of the candidate key which can uniquely identified records of a table.
- It is used by database designer to impose restriction on the table.
- eg - From the above set of candidate keys the best possible option is student id. So id will be made as the primary key of the table.

Alternate key -

- Candidate keys other than primary key which can uniquely identify a record from the table is called as alternate keys.

eg - {phone} {email-id}

Example -



Super key -

I: {eid} {name} {phno}

II: {eid, name} {eid, phno}, {name, phone}

III: {eid, name, phno}

Candidate key - {eid} {phno}

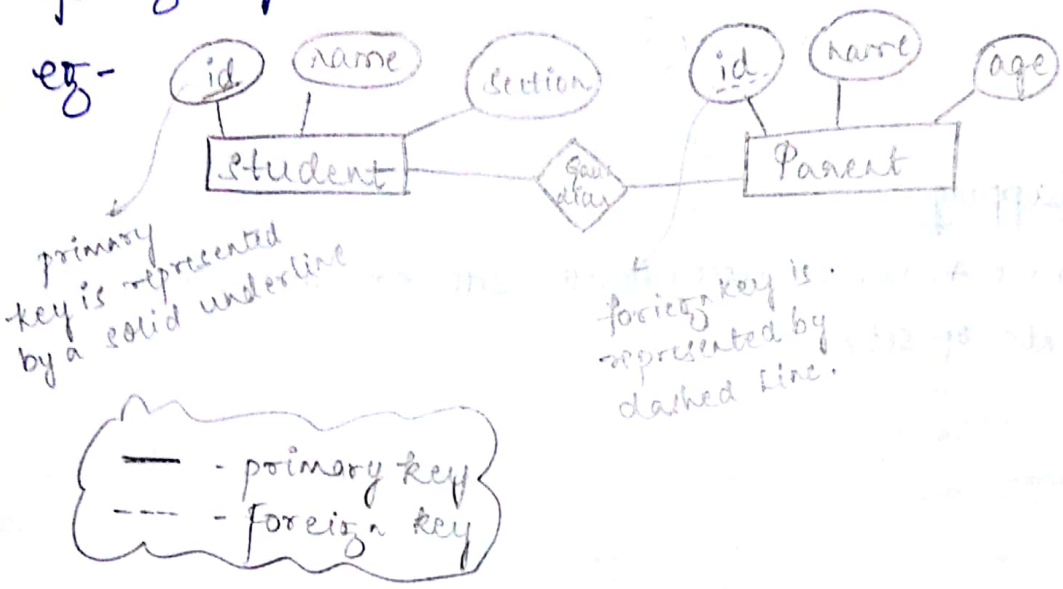
Primary key - {eid}

Alternate key - {phone no}

Foreign key -

If attributes of a particular table refers to primary key of some other table then that attribute will be called as foreign key.

eg -

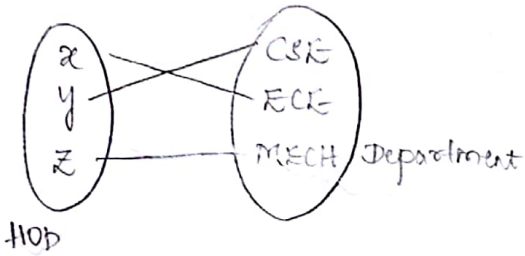


Mapping Cardinality -

Total number of relations a particular entity associates with some other entity is called as cardinality.

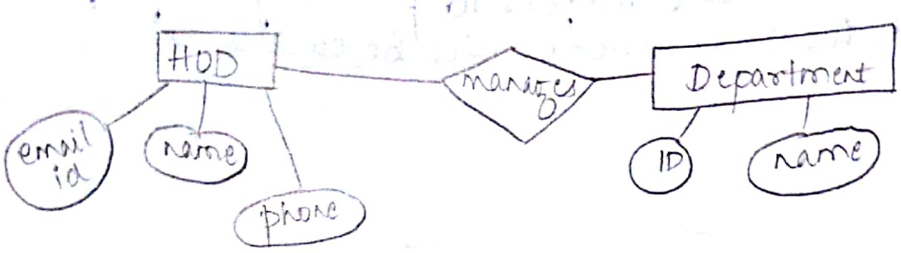
→ One-one -

eg -



Only one element of set A can associate with only one element of set B.

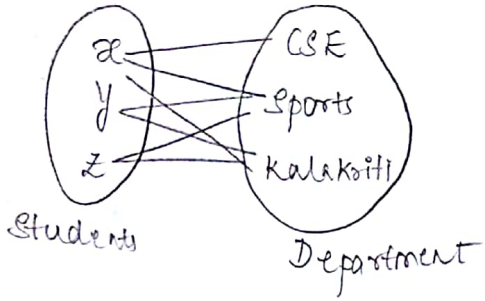
ER-diagram –



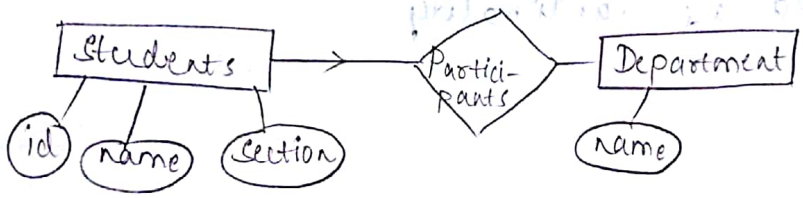
One to many Mapping :

One element in set A can be associated with one or more number of elements of set B.

eg –

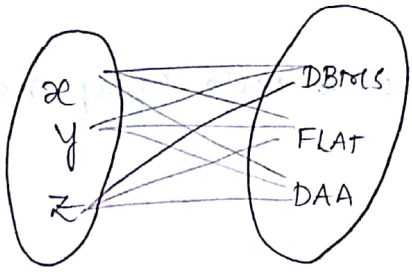


ER-diagram –



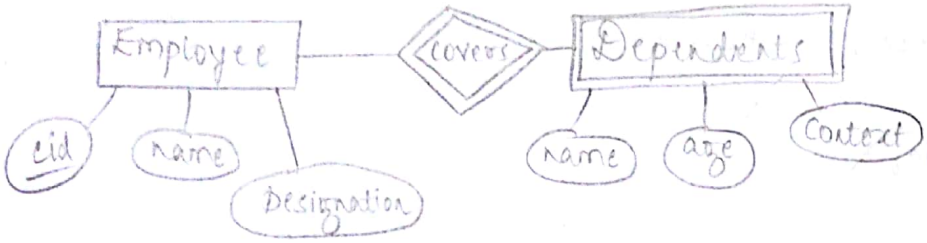
Many to many Mapping :

eg –



Weak Entity -

If an entity doesn't have its own key attribute then it is called as a weak entity.



- From the above diagram, 'Dependents' its a weak entity as it doesn't have its own key attributes.
- The other name for weak entity is 'Existential Dependent'.
- Strong Entity is also called as 'Identifying Entity' or 'Owner'.

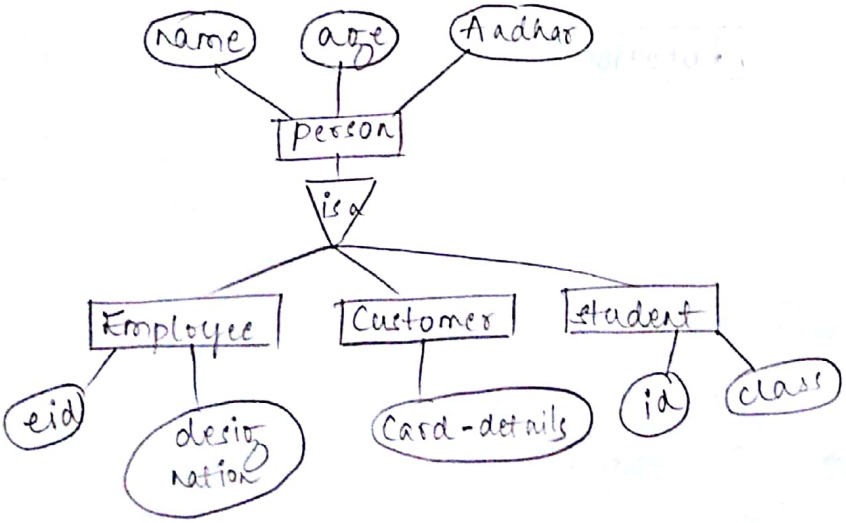
Generalization -

If a set of Entities are grouped together by adding a higher level entity as parent then it is called as Generalization.

- 'isa' is the relation used to connect base entity with child Entities.

For eg -

We have three entities student, Customer and employee they can be grouped together as follows:

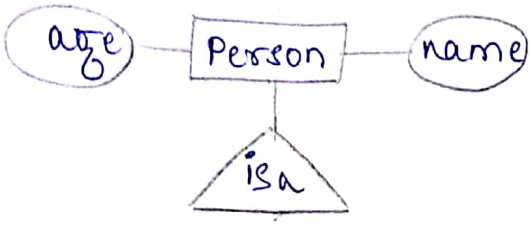


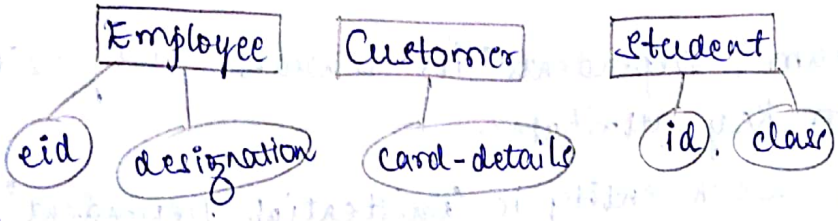
Specialization -

It is opposite to generalization.

If a higher level entity is categorised into a set of lower level entities, then it is called as specialization

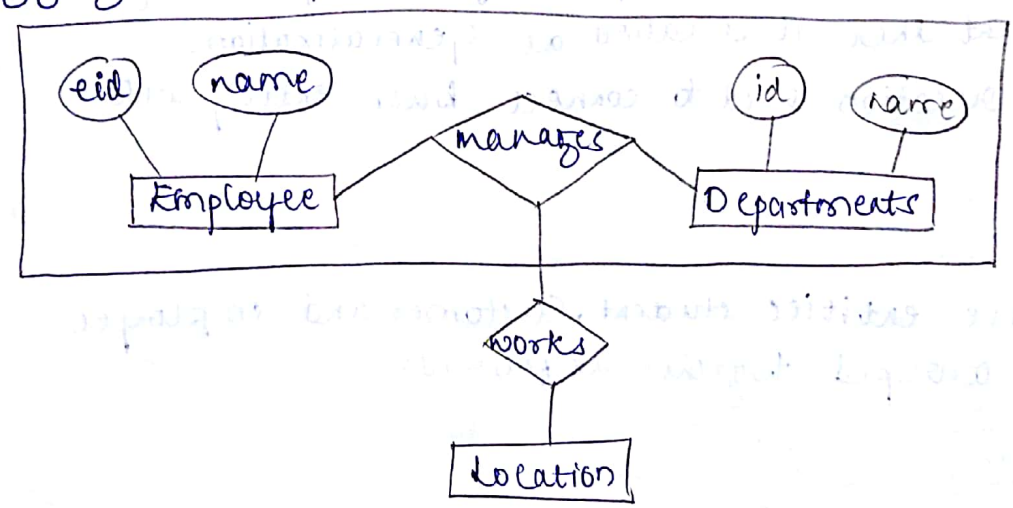
eg -





Aggregation -

To show a relation over another relation we use aggregation

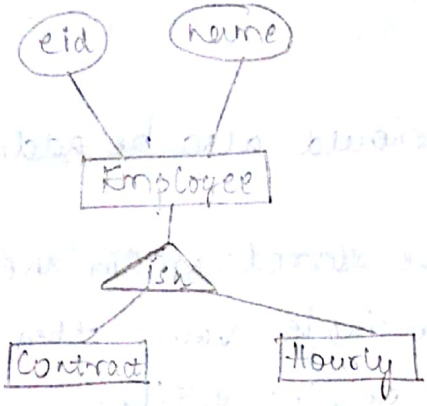


⇒ Is a Hierarchy –

Covering constraints :

All the low level Entities can together fulfill parent Entity (or) high level Entity then it is called as covering Constraints.

Eg –



- In contract & hourly employees together with fullfil the high level entity then we call it as "Contract & Hourly Employees covers Entity'.

Overlapping Constraints –

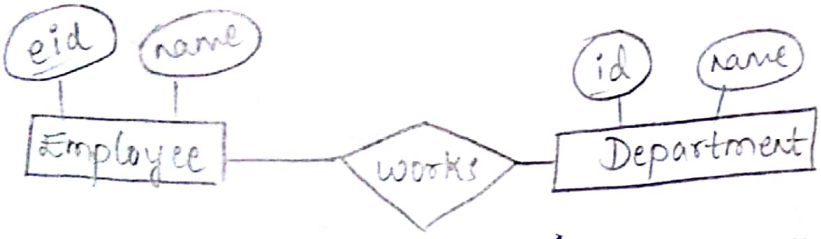
If all the hourly ~~paid then we call it as~~ Employees are also ~~caught~~ Contract Employees then we call it as "Hourly Employees" overlaps Contract Employee's.

- If all the contract Employees are not hourly paid then we call it as 'Contract Employees donot overlap hourly employees".

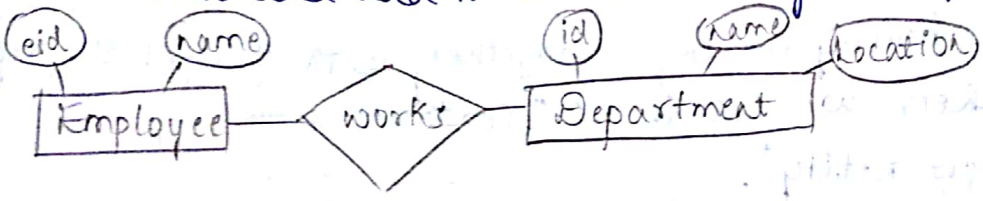
12.07.2017

* Conceptual Design with ER Diagrams

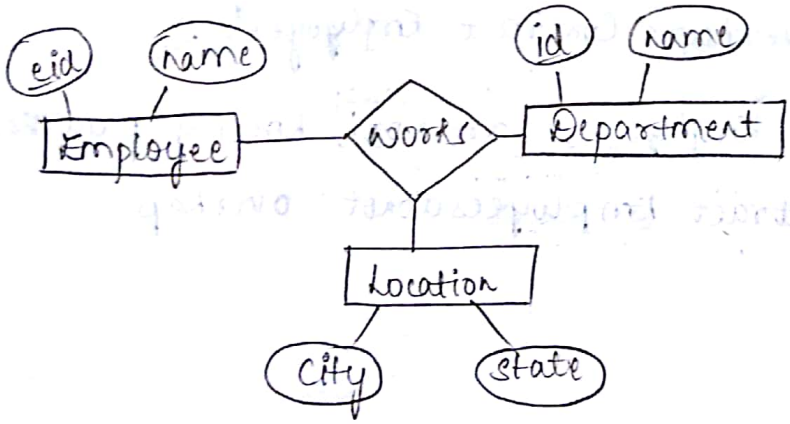
(1) Entity (vs) Attribute -



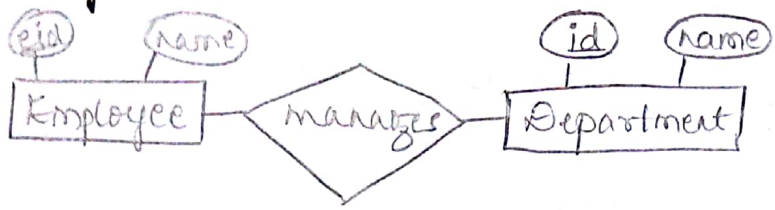
- If a data item called 'location' should also be added to the above ER-diagram.
  We should identify that will be stored within the data item.
- If the data item will store a single value then make it as an attribute and add it to the existing entity.



- If the data element is going to store set of values then, make it as an entity and relate it with an existing ER-diagram.



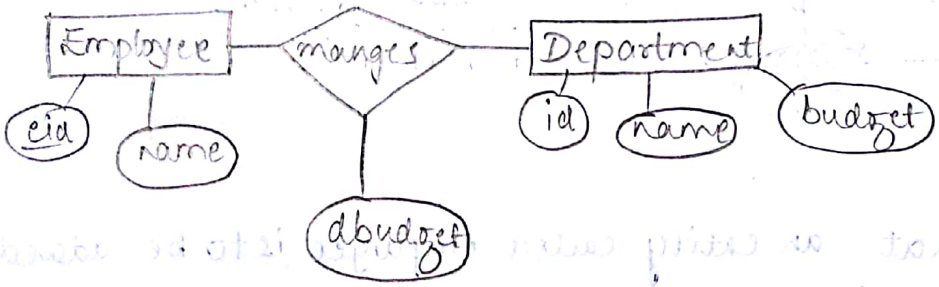
(2) Entity (vs) Relation -



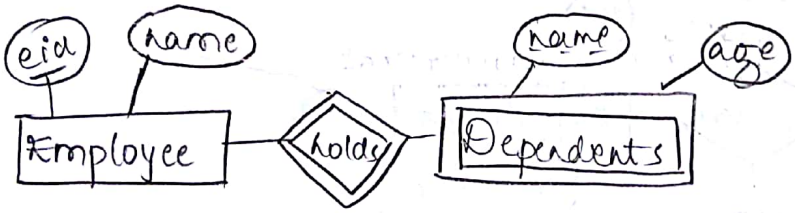
Consider that two types of budgets are given at organizational level.

(i) Budget - Budget allocated to every department will be sanctioned certain amount to perform interdepartment activities

(ii) discritionary Budget (dbudget) - This is the budget allocated to each manager to perform various activities that are not even inter departmental.

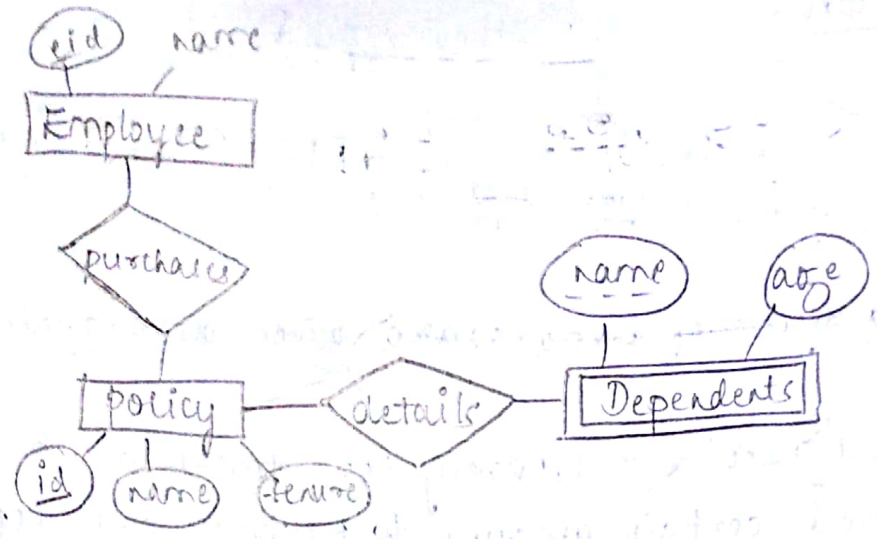


(3) Binary (vs) Terenary -

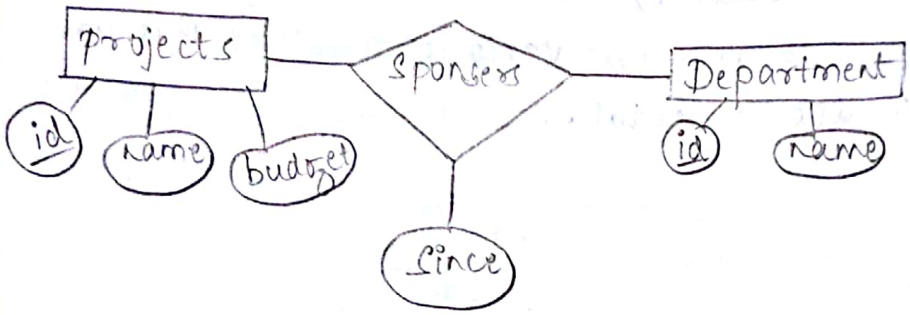


If an entity called 'policy' having attributes 'id', 'name', 'tenyr' is to be added to above relation - it is not possible.

As the above relation is a identifying relation or weak relation so the above E-R diagram will be reconstructed as shown below.

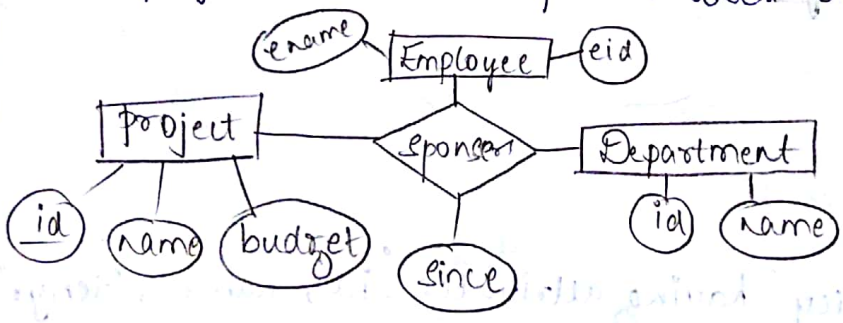


(4) Ternary (vs) Aggregation -

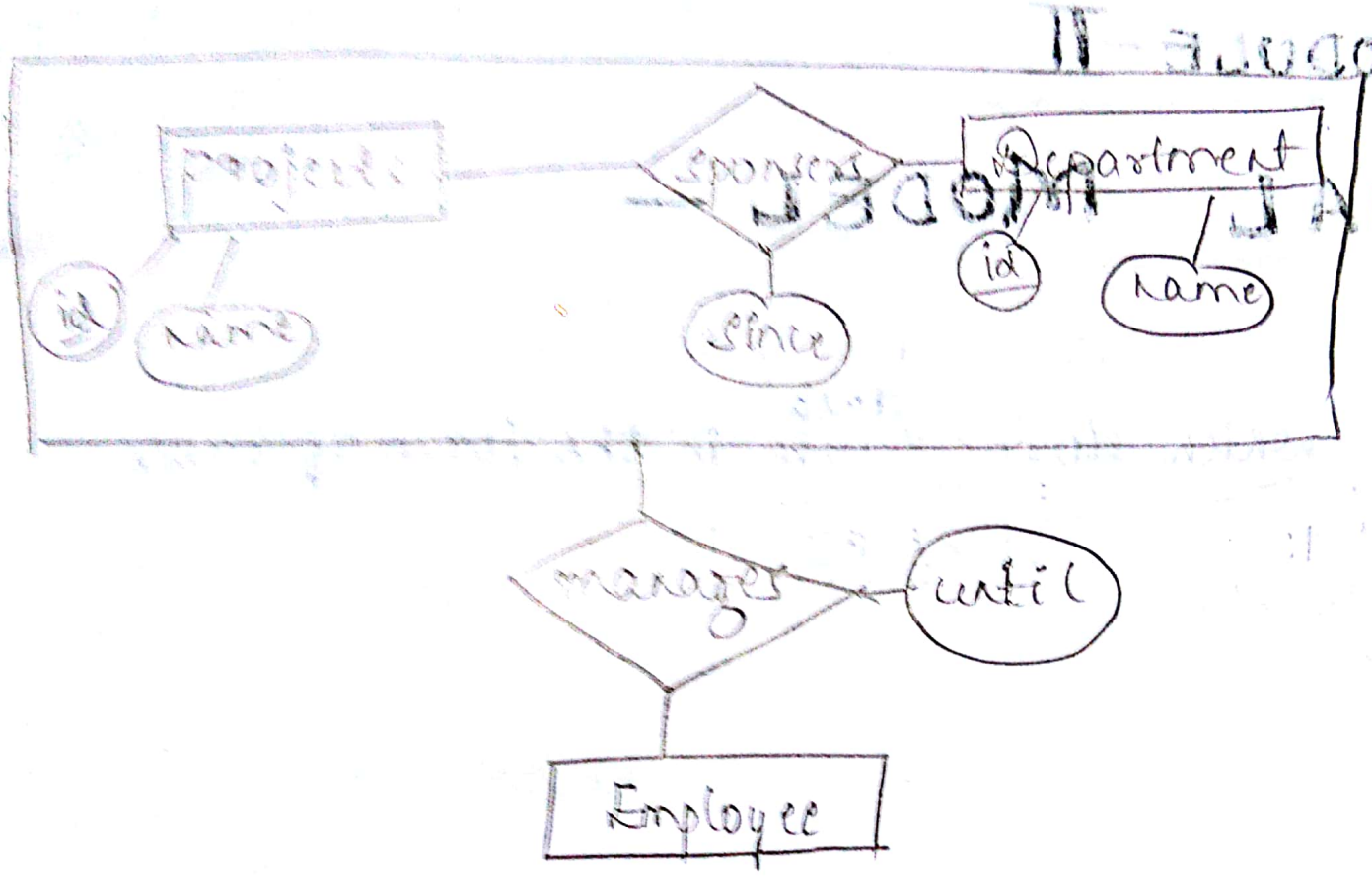


Consider that an entity called employee is to be added to the above relation.

Cond.1) If the same employee is responsible for managing the entire project until its completion then we get ternary relation.



Cond.2) If the employee is responsible for managing the project only for certain period of time. Then,

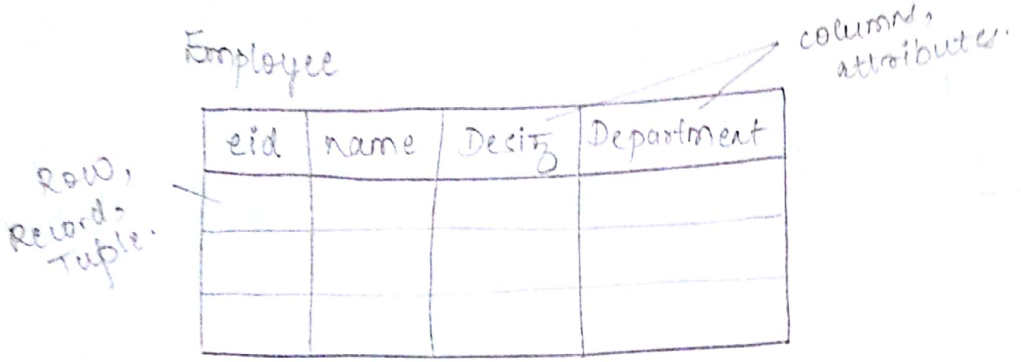


MODULE-II

— RELATIONAL MODEL —

Relational model -

It is the database which stores ~~data~~ table in the form of table



Relational model = Relational schema + Relational Instance.

where Relational schema is overall structure of table

Relational Instance is values stored in table at particular point of time.

Integrity Constraints -

Rules that are imposed on a table to organize data properly are called as integrity constraints.

1. Domain constraints
2. key constraints
3. NOT NULL
4. Check.

(1.) Domain Constraints -

- Domain - The set of all possible values accepted by column of a table is called as domain.
- Restricting each column to accept only paticular set of values is called as domain constraint.

eg - Student id - number (domain constraint) - It accepts only numericals.

(2) key Constraints -

unit-1
1. super key
2. Candidate key.
3. primary key:-
4) Unique
5) Foreign key.

3 Primary key -

It is an attribute or set of attributes which can uniquely identify a record from the table.

Rules:-

(i) primary key will never accept null value.
(ii) primary key never accept duplicate values
(iii) ~~Only~~ atmost only one primary key can be declared within a table
(iv) Primary key should be small and simple as possible.

Creation of primary key -

employee

eid	name	age	Location

at column Level -

(1) create table <table-name> (<col1> <type> primary key, <col2> <type>...);

eg - create table employee (eid number primary key, name char, age number, location varchar);

at table Level -

(2.) create table <table name> (<col1> type, <col2> <type> ... primary key(col1));

eg - create table employee (eid number, name char, age number, location varchar, primary key (eid));

adding primary key already existing table.

⇒ add -

alter table <table_name> add primary key (<col>);

eg -

alter table employee add primary key (eid);

⇒ drop primary key -

alter table <table_name> drop primary key;

eg -

alter table employee drop primary key;

14.07.2017

4. Unique Constraint -

˘ This constraint also helps to uniquely identify a record from the table.

Rules -

(i) A table can have any number of unique constraints unlike primary key.
(ii) It doesn't accept duplicate values
(iii) Unique can accept NULL values (only single null value for entire column).

˘ Similar to primary key unique constraint can be applied on two or more columns together.

Syntax -

create table <table-name>(<col1><type>,<col2> <type> ... unique(col));

(Or)

create table <table-name>(<col1> <type> ..... unique(col1,col2...));
two columns will be compared together.

(Or)

Create table <table-name>(<col1><type>.... unique(col1), unique(col2));
two columns will be compared seperately

eg-

Employee

eid	name	age	location	phone.

create table employee (eid number(10), name varchar(10), age number(10), location varchar(10), primary key (eid), unique(phone));

adding unique constraint to already existing table.

Syntax-

alter table <table-name> add Constraint <constraint_name> unique (col...);

eg- alter table employee add Constraint Const_employee unique (name);

Note - Name of the constraint should not match with existing table name or column names.

drop unique constraint to allready existing table-

Syntax-

alter table <t-name> drop Constraint <constraint-name>;

eg-

alter table employee. drop constraint const-employee;

We use unique ~~key~~ constraint when a table already has primary key and we also want to put constraints over other columns similar to primary key we make that column as unique constraint

5.

→ Foreign Key –

syntax –

create table <table name> (<col1> <type>, <col2> <type> ....,
primary key (col1), foreign key (col2) references <primarytable(pk)>

* The ~~p~~table that consists of primary key is called base table

project

Empid	pid	pname	budget
101	P1		
102	P2		
103	P3		

Reference table

Employee

eid	name	age	location	phone
101	xx			
102	yy			
103	zz			

base table

eg –

create table projects (eid number(10), pid varchar(10),
phone number(10), budget number(50), primary key (pid),
Foreign (Empid) references Employee (eid));

Employee
Dependent

- Adding Foreign key to already existing table.

add ⇒

alter table <t_name> add Constraint <const_name> Foreign key (col) references <Primarykey_table_name> (pk));

- drop foreign key from already existing table

alter table <t_name> drop Constraint <const_name>;

Foreign key definition -

If an attribute or set of attributes of a particular table refers to the primary key of some other table then this attribute or set of attributes will become foreign key of that table.

Rules -

(i) Records cannot be entered into Foreign key column until the same values are entered in the primary key table.

(ii) It can also accept duplicate values.

(iii) It can accept null values (leaving foriegn key record empty & can enter remaining details of that record).

(iv) The data types of foreign key column and primary ~~and primary key~~ key columns must be similar

(v) The records must be droped or deleted from the foriegn key table and then only we can drop or delete from primary key table.

(3.) CHECK CONSTRAINT -

- It restricts a particular attribute or set of attributes to accept values only with in the specified range.

Syntax -

Create table <table_name> (<col1><type>, <col2><type> ... PK FK UK ,

Check (condition));

Condition -

Col < value and col > value

Col != value

Col1 <= value and col >= value

so on ... like or, and, not,

<, >, <=, >=, =, != ...

In order to restrict col eid to accept particular range of values, then

eg - check (eid <= 66);

check (eid >= 1 and eid <= 66);

- A table can have any number of check constraints.
- Here the restriction is the check constraint can~~not~~ be only imposed over a column if that column type is numbers only.

(4.) NOT NULL -

- To restrict a particular column of a table not to accept null values.

Rules

- Here we cannot put restriction over primary key column
- any number of NOT NULL constraints can be present within a table

Syntax -

Create table <t_name> (<col1><type> NOT NULL, ...);

eg -

Create table employee (eid number(10), name varchar(10) NOT NULL);

25/07/2017

Enforcing Integrity Constraints-

1. Insertion
   - Parent - No restriction
   - Child - Restriction
2. updation
   - Parent - Restriction
   - Child - Restriction
3. deletion
   - parent - Restriction
   - Child - No restriction

- In order to violate the restrictions imposed on primary key table during updation and deletion we use the following terms

- No action

- No action will be performed if modifications are made in the parent table in child table

- Cascade -

- Changes made is parent table will be reflected in the child table.

- Set Null -

- changes made is parent table will reflect null values in child table.

- default <value>

- any changes are made in parent table child table will be reflected with default given value.

Student

id	name	section

parent table

Course

cid	cid	name

child table.

No action :-

- Create table student(id number(10), name varchar(10), section varchar(10), primary key(id));
- Create table course(sid number(10), cid varchar(10), cname varchar(20), constraint course_const Foreign key(sid) references student(id));

⇒ Delete-

Cascade

Create table Course (sid number(10), cid varchar(10), cname varchar(20), constraint course_const Foreign key(sid) refrences student(id) on delete cascade);

On delete set null-

Create table course (sid number(10), cid varchar(10), cname varchar(20), constraint course_const Foreign key(sid) references student(id) on delete set null);

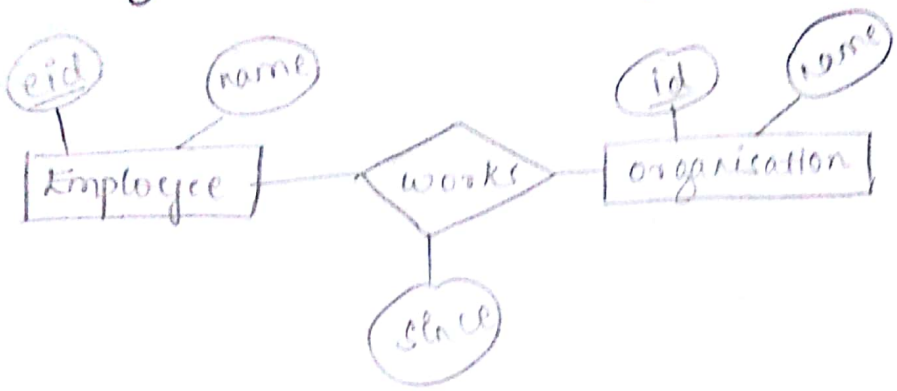
On delete default <value> -

create table course (sid number(10), cid varchar(10), cname varchar(20), constraint course_const Foreign key(sid) references student(id) on delete default <value>);

⇒ Update

- The same is for update also, in place of delete we replace with update

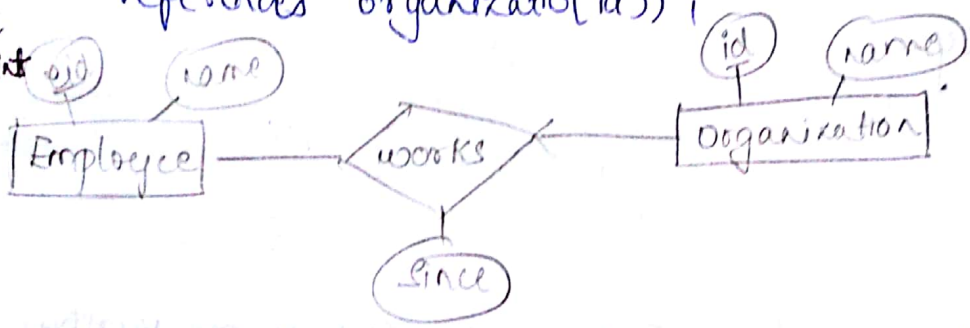
Converting Relations into query-



(i) Without constraints-

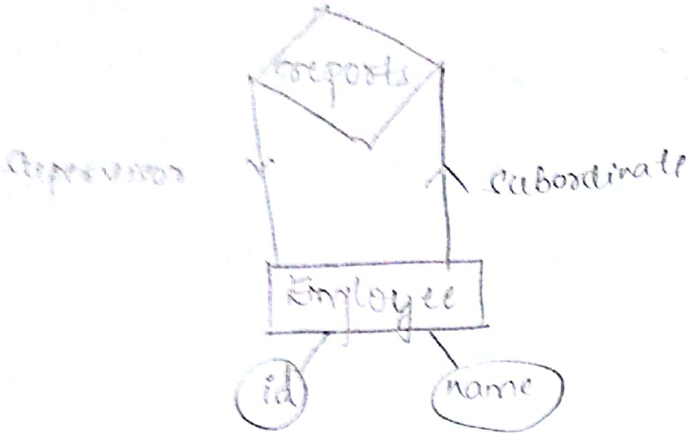
Create table works (eid num(10), id number(10), since date, primary key (eid, id), Foreign key (eid). references employee(eid) Foreign key (eid) references organizatio(id));

(ii) With constraint



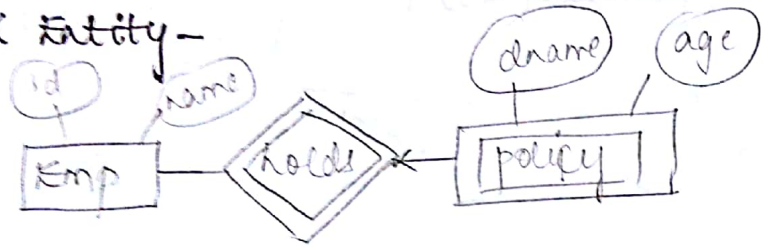
Create table works (eid number (10), id number(10), since date, primary key (id) foreign key (eid) references employee (eid) foreign key (id) references organisation (id));

(iii) Binary Relation without constraint.



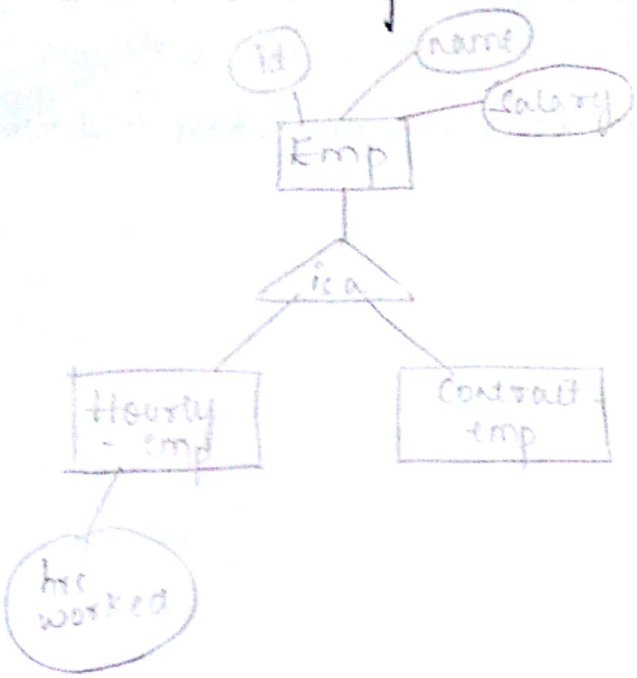
Create table works (sup_id number(10), sub_id number(10), since date, primary key (sup_id, sub_id) Foreign key (sup_id) references employee (sup_id) Foreign key (sub_id) refferences organization (sub_id));

(iv) Weak Entity -



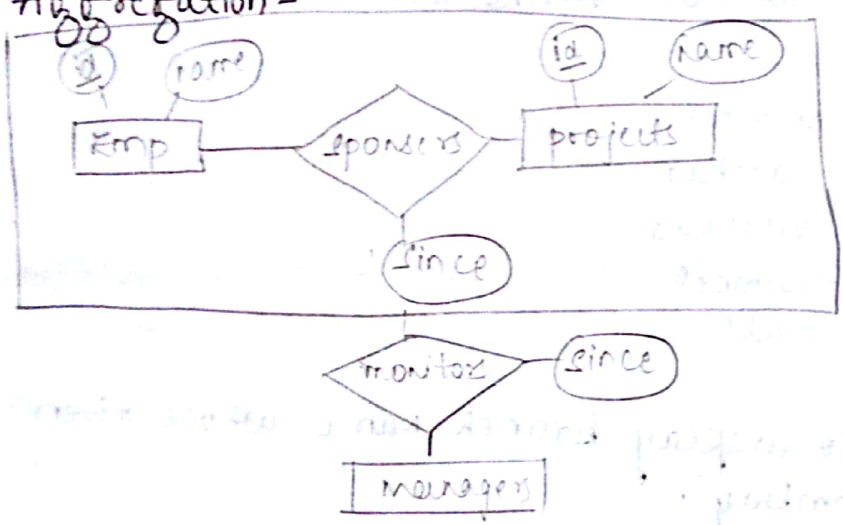
Create table works (id varchar(10), dname varchar(10), Sincedate, primary key (id, dname) Foreign key (id) references employee (id) Foreign key (dname) references policy (dname) on delete cascade);

(v) Is a Hierarchy -



create table hourly-emp (id number(10), hours_worked number(
primary key (id, hours_worked) Foreign key (id) references
Employee (id));

(vi) Aggregation -



(i) Key attribute of employee, key att
descriptive attribute of sponsers
+
(ii) descriptive attributes of monitors.
+
(iii) key attributes of managers.

27.07.2017

Querying Relational Data

Select distinct <col list> from <table-name> where condition;

distinct (optional) — where the duplicate values can be eliminated

where condition (optional)

Example-

I: Branch - bname : varchar
bcity : varchar

II. Customer - cname : varchar
ccity : varchar

III. Borrow - ~~Rollno~~ loanno : number
cname : varchar
bname : varchar
amount : ~~varchar~~ number

IV Deposit - Actno - number
cname : varchar
bname : varchar
amount - number
adate : date

Query

(1.) Write a query to display branch name whose given branch city = 'bombay'.

Select distinct bname from branch where bcity = 'Bombay';

(2) Write a query to display loan amount where given loan number = 100.
loan no = 100 and Amount.

Select amount from borrow where loanno = 100;

(3) Write a Query to display customer name and amount deposited in the account where given account number = 175

cname = ?
amount = ?
Actno = 175

Select cname, amount from Deposit where Actno = 175;

(4) Write a Query to display amount deposited in the account where account = 175 and Customer name = raju.

Amount = ? cname = raju

Select amount from deposit where Actno = 175 and cname = 'raju';

(5) Write a Query to display amount deposited in account where accountno = 175 and customer name = raju by eliminating duplicate value.

Select distinct amount from deposit where Actno = 175 and cname = 'raju';

Cartesian Join / Natural Join (⋈)

Simple Join.

This concept helps in joining two or more tables without using any additional key words.

Condition –

In order to join two tables using this method atleast one common column should exisit between the tables

eg- Write a Query to retrive cname when given branch city is bombay.

cname? bname = bombay.
(C, Bo, d) (B)
(Instances are necessary).

Select d.cname from deposit d, branch b where d.bname = b.bname, b city = bombay;

⑫ Write a query to retrive customer name where given bcity = 'delhi'.

```
Select Bo.cname from borrow bo, branch b where
bo.bname = b.name and b.bcity = delhi ;.
```

11th query

Write a Query to retrive actno and cname where given loan amount > 2000.

```
Select d.actno, d.cname from deposit d, borrow bo
where d.cname = bo.cname and bo.amount > 2000;
```

12th query

Write a query to retrive names of customer having city bombay and bcity = delhi

```
Cname=?   (D, Bo, C)
Lcity= Bombay (C)
bcity = delhi (B)
```

while the two reference table.

```
Select d.cname from Customerc, branch b, deposit d
    where d.cname = c.cname and d.bname = b.bname
and  c.city = 'bombay' and b.bcity = 'delhi';
```

Note -

In order to join 'n' number of tables using cartesian product we need to specify n-1 joining conditions.

13 Write a query to retrive names of customers whose living city = branch city.

```
Cname=? (D, Bo, C
lcity = bcity.
 (C)   (B).
```

deposit is the reference hence d.cname

```
Select d.cname from customer c, branch b, deposit d
where d.cname = c.cname and d.bname = b.bname and
    c.city = bcity;
```

14. Write a query to retrive details of 'cname' and 'adate' of customers where bcity = 'delhi' and loanno = 481.

cname = ? (D, R, C)
adate = ? (D)
loanno = 481 (BO)
bcity = 'delhi' (B)

Select d.cname and, d.adate from depasit d, borroco bo, branch b where d.cname = bo.cname and d.bname = b.bname and bo.loanno = 481 and b.bcity = 'delhi'.

15. Write a query to retrive. Living city (lcity) of 'anil' and 'sunil'.

Select city from Customer where cname = 'anil' and cname = 'sunil';

city = ? (c)
cname = anil (c)
cname = sunil (c).
(since all are of same table all can written with or without instances).

16 Write a Query to retrive lcity of 'anil' or 'sunil'.

Select city from customer where cname = 'anil' or cname = 'sunil';

(OR)

Select c.city from customer c where c.cname = 'anil' or c.cname = 'sunil';

01.08.2017

Views (or) Viewtable.

- It is a table created on the database which contain rows that will not be explicitly stored within the database.
- There are two types of view table
  (i) simple view.
  (ii) complex view.

(i) Simple view -

If a view table is created by adding columns of single base table then it is called as single base (view) table.

ex- student

ID	Name	Year	section	phno	email	mobile ....
1	Ramu	3	D			
2	Somu	3	D			
3	Anu	3	D			

Syntax for creating view table -

create view <view name> (col1, col2, ...) as
select <col1>, <col2> ..... from <tablename> where <condition>;

(The name that came be anything that will be visible in viewtable.)
(These names should be as same as base table)

eg-

create view view_student1 (ID, Name, Year, section)
as select ID, name, year, section from student;

(OR)

with instances

create view view_student1 (ID, Name, year, section)
as select s.ID, s.Name, s.year, s.section
from student s;

ex2

Course

sid	CID	cName	Credits	Year	.....

Write a Query to create a view table by adding colums CID and CNAME.

Create view view-cource (CID, CNAME) as select C.CID, C.CName from Course C;

(ii) Complex View -

- If a view table is created by adding columns of two or more base tables then it is complex view.

Syntax

Create view <viewname> (<col1>, <col2>, ...) as select <col1>, <col2> .... from <table1> <table2> ... where <condition>

Q Write a Query to create view table by adding colums SID, name, CID, CNAME from student and course tables.

Create view view-stco (SID, NAME, cid, cname) as select S.SID, S.Name, C.cid, C.Cname from student S, course C where S.ID = C.sid;

↳ If columns name are to be same with datatypes are to be similar. But their should be joining condition.

DML Operations of simple view

(i) insert -

Base Table - Any number of records can be inserted into base table irrespective of view tables created on it.

View Table - A new record can be inserted into view table only if the view table contains primary key of base table.

Eg- view_student1 table a new record can be inserted as view table contains primary key of base table.

Here when we insert into base tabl that will be reflected into view table and also vice versa.

(ii) Delete -

Base Table - Any number of records can be deleted from base table without any restrictions.

View table - Records can be deleted from view table provided the view table should contain primary key of base table.

(iii) Update -

Base Table - Any number of records can be updated from base table without any restrictions

View Table - Records can be updated from view table provided the view table should contain primary key of base table.

DML operations on Complex View –

(i) Insert and Delete –

Insert and delete operations can be performed on complex view table provided view table should contain primary keys of all the base tables.

{Any modifications made in view tables changes will be made for respective base tables and vice versa.}

(ii) Update –

- Update operation cannot be performed on complex view table even though the view table contains primary keys of base table. This is called as update restriction over complex view.

(If asked for 5m or 12m - we need to write complete complex and simple view)

Drop a View table –

In order to drop an entire view table

drop view <view name>;

Alter View table –

alter view <view name> drop column <col-name>;

⇒ RELATIONAL ALGEBRA-

Date:- 03.08.2017

-It is a procedural Query language which results in a single relational value by combining two or more relational values.

(i) (σ)- Selection

In order to retrive a record from the table when a condition is given.

Example-

Sailor(S)

Sid	name	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.0
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horaito	7	35.0
71	zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserved(R)

Sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats(B)

Bid	bname	Colour
101	Interlake	blue
102	Interlake	Red
103	clipper	green
104	Marine	red.

Syntax- selection

σ (Table_name) condition

(OR)

σ (instance) condition

(ii) Projection (π)

In order to retrive columns from the given table we use projection

Syntax

$\pi_{col\_list}(Table\_name)$

(OR)

$\pi_{col\_list}(Table\_instance)$

Q. Write a Query to display details of sailors whose rating is > 7.

$\sigma_{rating>7}^{(Sailors)}$ (OR) $\sigma_{rating>7}(S)$

Q. Write a Query to display names of sailors.

$\pi_{name}^{(Sailors)}$ (OR) $\pi_{name}(S)$

Q. Write a Query to display name and age of sailors

$\pi_{name,age}(Sailors)$

Q. Write a Query to display names of sailors whose rating > 7.

rating > 7, name = ?

$\pi_{name}(\sigma_{rating}^{(S)})$

Q.) Write a Query to display sid and names of sailors whose age is between 25 to 35.

age < 35, age ≥ 25 name = ?

$\pi_{sid,name}(\sigma_{age\geq 25 \text{ and } age<35}^{(S)})$

Q) display id of sailors where sailor name = Rubber.

$\pi_{sid}(\sigma_{name="Rubber"})$

(iii) Set Operations -

(i) Union (∪)

If R and S are two relations then they can be combined using set operations only if they are 'union compatible'.

Union compatibility -

Two tables are said to be union compatible if the following conditions

(a) Both the tables should have equal number of columns.

(b) Starting from left to right columns of both the tables should be similar.

R ∪ S Here two relations R and S will be combined together to form a single relation containing all columns of R and S.

(ii) Intersection (∩) -

- Two tables can be combined using intersection operation provided they are 'union compatibility'
- R ∩ S When two relations R and S are combined using intersection operation a single relation will be displayed by displaying common columns between R and S.

(iii) Set Difference (−) -

- We can perform set difference operation provided they are 'union compatible'
- R − S If R and S are two relations/tables then 'R − S',
  (a) Table 'R' will be compared with 'S' table, similar records will be eliminated from table R and ~~the~~.
  (b) remaining records of relation 'R' will be displayed in the result.

(iv) Joins (⋈) -

(i) Inner Join
- θ Join (conditional)
- Equi Join
- Natural Join

(ii) Outer Join
- Left Join
- Right Join
- Full Join

(i) Inner Join -

- Inner Join combines two relations/table and displays result when the given condition is satisfied.
- This inner join categorised into
  - θ Join (conditional Join)
  - Equi Join
  - Natural Join

(a) Theta Join -

- If R and S are two relations/tables we can join them only if the given condition is satisfied.
- It is symbolically represented as $R \bowtie_{R.col > S.col} S$ $=, >, <, <=, >=, !=$ etc.

(b) Equi Join -

- If R and S are two tables when both the columns are equal.
- symbolically represented as $R \bowtie_{R.col = S.col} S$

→ Here columns including values to be equal in the given column in the condition

(ii) Outer Join

(a) Left outer Join ⟕

If R & S are two relations then R ⟕ S displays common values present in both the tables followed by instances ~~in both~~ of the ~~tables~~ left hand side table (R) will be displayed.

Example -

student

sid	cid
1	101
2	103

Course

cid	cname
101	XX
102	YY
104	ZZ

Sid	cid	cname
1	101	XX
2	103	YY

(b.) Right Outer Join (⟖)

If R and S are two relations then R⟖S displays common values present in both the tables followed by instances of Right hand side table.

eg-

Sid	Cid	Cname
1	101	XX
2	102	YY
	104	

(c.) Full outer Join (⟗)

If R and S are two relations then R⟗S displays common values present in both the tables followed by instances of both the tables.

eg-

Sid	Cid	Cname
1	101	XX
2	103	
	102	YY
	104	

Rename (ρ)

This operator helps in renaming a set of relations with a temporary name

eg- $\rho(x, y)$ y will be replaced ~~by~~ with x.

→ Division (/) :-

- If R and S are two relations, R is a relation having two columns x and y. and s is a table having column y.
- R/S is given by <y> in "s" there should exists <x,y> in "R"

Ex-

R

x	y
A_1	P_1
A_1	P_2
A_3	P_3
A_4	P_1
A_5	P_1
A_6	P_2
P_7	P_3

S_1

y
P_1

$R/S_1 = A_1, A_4, A_5$

A_1
A_4
A_5

S_2

y
P_1
P_2

$R/S_2 = A_1$

A_1

S_3

y
P_1
P_2
P_3

$R/S_3 = NULL$ □

Example Queries-

(1.) Write a Query to retrive names of sailors who reserved ~~boat 1,0,3~~ boat 103.

π_{sname} = sailors

$\sigma_{bid=103}$ = Reserves.

$$\pi_{s.sname}\left(\left(\sigma_{R.bid=103} R\right) \bowtie S\right)$$

(2.) Write a Query to display names of sailors who reserved boat on 10/10/98

π_{sname} = sailors

$\sigma_{day = 10/10/98}$.

$$\pi_{s.sname}\left(\left(\sigma_{R.day='10/10/95'} R\right) \bowtie S\right)$$

(3.) Write a Query to retrive names of sailors who reserved red colour boat.

π_{sname} = sailors

$\sigma_{sname = red}$.

As there is no common column between boats and sailors we take reserves table as reference to join sailors and boats table

$$\pi_{s.sname}\left(\left(\sigma_{B.colour='red'}(B) \bowtie R\right) \bowtie S\right)$$

(4) Write a Query to display names of sailors who reserved red colour boat by using rename operator.

Step 1 : $\sigma_{B.colour='red'}(B)$

step 2 : $\rho\left(Temp1, \sigma_{B.color='red'}(B)\right)$

step 3 : $\pi_{S.sname}\left((Temp1 \bowtie R) \bowtie S\right)$

(5) Write a Query to display colour of boat reserved by sailor Lubber.

$\pi_{color}(B) = ?$ \ common column b/t boats and sailors we use reserves as reference.

$\sigma_{sname='lubber'}(S)$

step -1 $\sigma_{S.sname='lubber'}(S)$

step 2 $\rho\left(Temp, \sigma_{S.sname='lubber'}(S)\right)$

step3 $\pi_{B.color}\left((Temp \bowtie R) \bowtie B\right)$

(6) Find names of sailors who reserved red or green color boats

$\pi_{name}(S)$

$\sigma_{B.color='red' \vee 'green'}$

$\rho\left(Temp, \sigma_{B.color='red' \vee 'green'}(B)\right)$

$\Rightarrow \boxed{\pi_{S.sname}\left((Temp \bowtie R) \bowtie S\right)}$

(7) Write a Query to retrive names of sailors who reserve at least one boat.

$\pi_{sname}(S)$

$$\pi_{s.sname}\left(\sigma_{s.sid=R.sid}(R \bowtie S)\right)$$

(8) Write a Query to retrive names of sailors who reserve all boats

$$\rho\left(Temp, \pi_{sid,bid}(R) / \pi_{bid}(B)\right)$$

$$\Rightarrow \boxed{\pi_{s.sname}(S \bowtie Temp)}$$

(9) Write a Query to retrive names of all sailors who reserved all boats called interlake

$$\pi_{s.sname}\left(S \bowtie \left(\pi_{sid,bid}(R) / \pi_{bid}\left(\sigma_{b.bname='interlake'}(B)\right)\right)\right)$$

(10) Write a Query to retrive sid of sailors whose age is greater than 20 and who have not reserved red coloured boat.

$\Rightarrow$ All color boats + age > 20 − red color boats

π

$$\pi_{s.sid}\left(\sigma_{s.age>20}(S) \bowtie R\right)$$

$$\pi_{s.sid}\left(\left(\sigma_{B.bcolor='red'}(B)\right) \bowtie R \bowtie S\right)$$

Ans

$$\pi_{s.sid}\left(\sigma_{s.age>20}(S) \bowtie R\right) - \pi_{s.sid}\left(\left(\sigma_{B.color='red'}(B)\right) \bowtie R \bowtie S\right)$$

(11) Write a Query to retrive sid of sailors whose age is greater than 20 and reserved red coloured boat.

$\pi_{sid}(R)$

$\sigma_{boat\,colour='red'}(B)$; $\sigma_{age>20}(S)$

$\Rightarrow \sigma_{B.color='red'}(B)$

$\Rightarrow \rho\left(Temp, \left(\sigma_{B.color='red'}(B)\right) \bowtie R\right)$

$\Rightarrow \pi_{s.sid}\left(Temp \bowtie \pi_{s.sid}\left(\sigma_{s.age>20}(S)\right)\right)$

12.08.2017

RELATIONAL CALCULUS

It is a non procedural language unlike relational algebra.
It is divided into two categories

(i) Tuple relational calculus
(ii) Domain relational calculus.

(i) Tuple Relational Calculus (TRC) -

Tuple relational calculus is represented as

$$T.R.C = \{ P / T(P) \}$$

where $P \rightarrow$ free variable

$T(P) \rightarrow$ formula applied on variable.

⇒ Formula for $T(P)$ -

1. Atomic formula

1.1 $R \in$ Relation

eg- $B \in$ Boats

1.2 $R.a$ op $s.b$

$R, s \rightarrow$ instances of relation
$a, b \rightarrow$ columns of relation
op $\rightarrow$ $=, <, >, <=, >= \ldots$

eg- $R.age < s.age$

1.3 $R.a$ op constant

eg- $R.age < 20$.

2. $\neg P$, $P \vee q$, $P \wedge q$, $P \Rightarrow q$ ....

3. ($\exists$ their exists) $\exists R$, $R \in$ relation.

4. $\forall$ (for all) R, $R \in$ Relation.

Queries

(1) Write a Query to retrive names of sailors whose rating is greater than 7.

name(S), rating < 7(S)

$\{P / \exists S \in sailor (S.rating < 7 \wedge P.name = S.name)\}$ → Relational calculus

Free variable

condition can be in any order

$\pi_{S.name}(\sigma_{rating<7}(S))$ → Relational algebra

(2) Write a Query to display reservation dates of sailors whose bid = 103.

day = ? (R) bid(R) = 103

$\{P / \exists R \in Reserves (R.bid = 103 \wedge P.day = R.day)\}$

(3) Write a Query to find name, ages of sailor whose rating > 7

$\{P / \exists S \in sailors (S.Rating > 7 \wedge P.name = S.name \wedge P.dage = S.age)\}$

(4) Find sailor name, boat id and reservation date for each reservation.

order is not mandatory

$\{P / \exists S \in sailors, R \in Reserves (R.sid = S.sid \wedge P.name = S.name \; P.day = R.day \wedge P.bid = R.bid)\}$

order is not mandatory within braces

(5) Write a Query to retrive names of sailors who have reserved red color boat.

$\{P / \exists S \in sailors, R \in Reserves, B \in Boats (B.color = 'red' \wedge P.bid = B.bid \wedge P.bid = R.bid \wedge P.sid = R.sid \wedge P.sid = S.sid \wedge P.name = S.name)\}$

(6) Find names of sailors who reserved all boats.

$\{P / \forall S \in sailor, R \in Reserves, B \in Boats (R.sid = S.sid \wedge R.bid = B.bid \wedge P.name = S.name)\}$

$\{P / \exists S \in sailors, \exists R \in Reserves (S.sid = R.sid \wedge P.name = S.name \wedge \forall B \in Boats (R.bid = B.bid))\}$

(7) Write a Query to display names of sailors who reserved all red color boats.

name = ?
∀ boats ; color = 'red' (B)

$\{P / \exists s \in sailors, \exists R \in Reserves (s.sid = R.sid \wedge P.name = s.name \wedge \forall B \in boats (R.bid = B.bid \wedge B.color = 'red'))\}$

(8) Write a Query to display color of boats reserved by sailor Lubber.

color = ? (B)
name = 'Lubber' (S)

$\{P / \exists B \in Boats, R \in Reserves (R.bid = B.bid \wedge P.color = B.color \wedge \exists S \in Sailors (s.name = 'Lubber' \wedge s.sid = R.sid))\}$.

Write a Query to display age of sailor who reserved green color boat.

P / ∃ .

Write a Query to display names of the boat reserved by sailor-" Lubber".

Domain Relational Calculus (DRC)

Sailors = I, N, R, A
Reserves = RI, BI, D
Boats = BD, BN, BC
} instances of columns.

$D.R.C = \{ \langle x_1, x_2 \dots x_n \rangle / P(x) \}$

where $x_1, x_2 \dots = x$ (Domain Variable)
$P(X)$ = Formula on Domain value.

(1) Write a Query to display names of sailors whose rating is greater than 7.

$\pi_{s.name}(\sigma_{s.rating>7}(S))$ (Relational Algebra)

D.R.C

$\{\langle N \rangle / \exists \langle I, R, A \rangle (\langle I, R, N, A \rangle \in sailors \wedge R > 7)\}$

(2) Write a Query to display sid of sailors who have reserved boat id = 103

boat id BI = 103 (R)
sid RI = ? (R)

$\{\langle RI \rangle / \exists \langle BI, D \rangle (\langle RI, BI, D \rangle \in Reserves \wedge BI = 103)\}$

(3) Write a Query to display names of sailors who reserved a boat, boat id = 102.

Boat id BI = 102 (R)
Sailor name N = ? (S)

$\{\langle N \rangle / \exists \langle I, R, A \rangle (\langle I, N, R, A \rangle \in sailors \wedge \{\exists (RI, BI, D) \in Reserves (BI = 102 \wedge I = RI)\})\}$

(4) Write a Query to display name of sailor and Reservation date of bid = 102.

Name N = ? (Sailors) BID = 102 (R)
R.date D = ? (R)

$\{\langle N, D \rangle / \exists \langle I, R, A \rangle (\langle I, N, R, A \rangle \in sailors \wedge \{\exists \langle RI, BI \rangle (\langle RI, BI, D \rangle \in Reserves (BI = 102 \wedge I = RI))\})\}$

(4) Write a Query to display names of sailors who reserved red color boat.

Boat color $BC = 'red'(B)$

$N(S) = ?$

$\{\langle N \rangle / \exists \langle I, R, A \rangle (\langle I, N, R, A \rangle \in sailors (\exists \langle RI, BI, D \rangle \in Reserves (I = RI (\exists \langle BD, BN, BC \rangle \in Boats (BI = BD \wedge BC = 'red')))))\}$

$\{\langle N \rangle / \exists \langle I, R, A \rangle (\langle I, N, R, A \rangle \in sailors \wedge \exists \langle RI, BI, D \rangle \in Reserves \wedge (I = RI \wedge \exists \langle BD, BN, BC \rangle \in Boats (BI = BD \wedge BC = 'red')))\}$

(5) Write a Query to display color of boats reserved by sailor Lubber

$\{\langle BC \rangle / \exists \langle BD, BN \rangle (\langle BD, BN, BC \rangle \in Boats \wedge \exists \langle RI, BI, D \rangle \in Reserves (BI = BD \wedge \exists \langle I, N, R, A \rangle \in sailors (I = RI \wedge N = 'Lubber')))\}$

MODULE-III

Date - 18.08.2017

SQL QUERIES and Advanced SQL

ADVANCED SQL-

SET OPERATIONS-

1. UNION-

When two or more relations are combined together to form a single relation by combining all the values of previous relations.

Syntax-

Select <col_list> from <table1> .... where <condition>
UNION select <col_list> from <table2> ... where <condition>;

Ex-

sailor1

sid	name	age

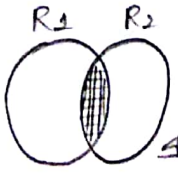
sailor2

sid	name	age.

select sid, name, age from sailor1 ~~where~~ union
select sid, name, age from sailor2;

2. Intersect -

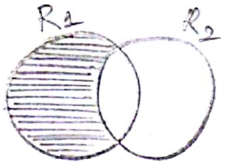
When two or more relations are combined together to form a single relation



Select <col_list> from <table1> ... where <condition>
intersect select <col_list from <table2> ... where <condition>;

(3.) Minus or Except -

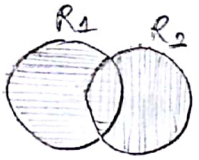
If R_1 and R_2 are two relations then R_1 minus R_2 $(R_1 - R_2)$ will display only the values of R_1 table/relation that do not match with R_2.



Syntax -

Select <col_list> from <table1> where <condition> minus
Select <col_list> from <table1> where <condition>;

(4.) Union all -



The difference between union and union all is, in the case of union set operations if any duplicate records are present within the table/relation it will be eliminated in final result But in case of union all setoperation, all the records will be displayed without elimination of duplicate records.

Syntax -

Select <col_list> from <table1> where <condition> unionall
Select <col_list> from <table2> where <condition>;

→ JOINS - ⋈ (Refer to unit-2)

- In outer Join, the two tables need not have similar columns or data present atleast one of the table or both the tables will be displayed as output.
- In the inner Join, the two tables must have common data in between them then only this join is possible otherwise no output is displayed.

1. Left Outer Join -

- select <col-list> from <left table1> leftjoin <right table> on <joining_condition> where <condition>
- Ex -

select s.sname from sailors s left join reserves R on R.sid=S.sid;

Q.) Write a Query to display names of sailors who reserved atleast one boat and also display details of all the sailors.

2. Right Outer Join -

- syntax -

select <col-list> from <left table> right join <right table> on <joining_condition> where <condition>;

- Ex -

Q) Write a Query to display names of sailors who reserved atleast one boat.

select s.sname from sailors s right join reserves R on R.sid=s.sid;

3. Full Join -

- syntax -

Select <col_list> from <left_table> full Join <right_table> on <joining_condition> where <condition>;

- Example -

Select * from sailors s full Join reserves R on R.sid=s.sid;

Inner Join -

- select <col_list> from <left table> inner join <right table> on <joining condition> where <condition> ;
- Example -

select * from sailors s inner join reserves R on R.sid = s.sid;

→ Null Values -

- create -
- create table <t_name> (<col1 type> NOT NULL, ...);

If any column name is followed by NOT NULL condition, it mean that the column is restricted by Null values.

- insert -

I : insert into <table> values ('value', ..., '');

II : Enter sid :↵

- update -

update <table> set col = 'value' where col is 'null';

- is, is not - In order to retrive any record refering to null value then we use above two operations.

Ex - student

sid	name	age
1	xx	19
2	yy	18
3		20

(1.) Write a Query to display details of students whose name is null

A. select * from student where name is 'null';

(2.) Write a Query to display sid of students whose name is not null.

A. select sid from student where name is not 'null';

- update -

update <table> set col = 'value' where col is 'null';

- delete -

delete * from student where col is 'null';

- orderby, desc -

→ SQL Functions -

Aggregate functions -

1. min() -
To identify minimum value of column
- syntax - select min(col) from <table_name>;

2. max() -
To identify maximum value for column
- syntax - select max(col) from <table_name>;

3. count(*) -
It counts number of values within a column by eliminating duplicate records.
- select count(col) from <table name>;
- select count(distinct col) from <table name>;

4. AVG -
- select avg(col) from tablename;
- To find the average of the values in a particular column.

5. sum -
- To find sum of values in a particular column.
- select sum(col) from table name;
- select sum(distinct col) from table name;

6. Group by, having -
- Write a query to display minimum age of sailors.

```
select min(age) sailors;
```

- Write a query to display minimum age of sailors for each rating level

eg

```
select min(age) from sailors group by rating;
```

```
select distinct <column list> from <table_list> where <condition> group by <col>;
```

• Write a Query to display minimum age of sailors for each rating level having rating level greater than 7.

```
select min (age) from sailors group by rating having
    rating >7 ;
```

• Write a Query to display minimum age of sailors for each rating level and age of sailor to be greater than 20.

```
Select min (age) from sailors where age>20 group by rating;
```

NESTED Queiries - (in, not in)

29.08.2017

If a Query is enclosed inside another Query then it is called as Nested Query

In order to write a Nested Query we use in and not in operations

Eg:- WAQ to display names of sailors whose boat id is 103

```
select s.name from sailor s, Reserves R where r.sid = s.sid and
                                              r.bid = 103;
```

Nested

```
select s.name from sailors s where s.sid in (select r.sid
        from reserves R where R.bid = 103);
```

Eg-2 WAQ to retrive sid of sailors who reserved red color boat

```
sid = ? (R)
color = red (B)   } Bid

select R.sid from Reserves R where R.bid in
    (select B.bid from Boats B where B.color = 'red');
```

Eg-3 WAQ to retrive names of sailors who reserved red color boat

```
name = ? (S)
color = red (B)
```

Reserves(R) for joining

```
Select S.name from sailors S, boat B, reserves R where r.bid = b.bid and
S.sid = r.sid and b.color = 'red';
```

⇒

```
(Select s.name from Sailors S where s.sid in
    (Select R.sid from Reserves R where R.bid in
        (select B.bid from Boats B where B.color = 'red')));
```

Eg-4 Write a Query to retrive names of sailor who have not reserved boats bid = 103.

```
select s.name from sailors S where s.sid not in
    (select r.sid from reserves R where R.bid = 103);
```

CORELATED QUERIES (exists)

- It is all a nested query where 'exists' is a keyword what we use. Corelated is mix of cartesion join & nested queries.

NAQ to retrive names of sailors who reserved boats with bid=103

```
Select s.sname from sailors s where exists
 (select * from reserves R where R.sid = s.sid and
  R.bid = 103);
```

- Correlated Queries combines features of both nested Queries and cartesian product (simple Join)
- As it Que contains one query nested inside another query similar to nested Query and...
  it contains joining condition similar to cartesian product.
- Exists is a key word in corelated Queries.

31.08.2017

⇒ **Any, All -**

(Q.) WAQ to retrive sid of sailors whose rating is greater than 7

```
Select s.sid from sailors s where rating > 10;
```

(Q) WAQ to retrive sid of sailor with rating > rating of 'Lubber'

(can be >, <, <=, >=)

```
Select s.sid from sailors s where s.rating > any (select
 s.rating from sailors s where s.sname = 'Lubber');
```

Any — example

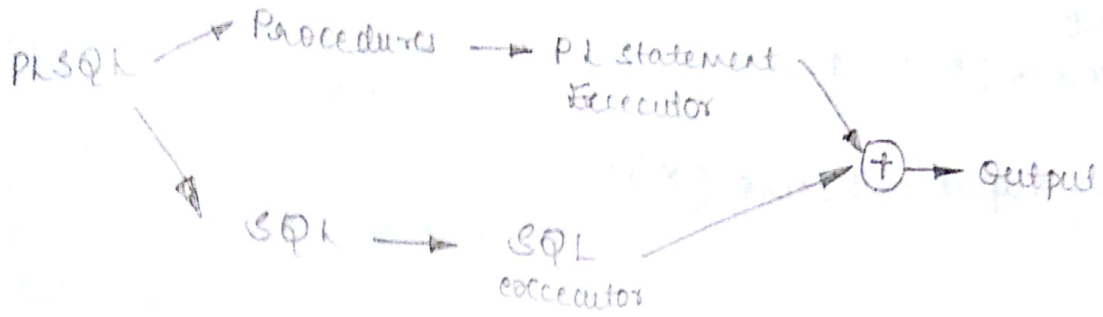
This Any key word will be used in nested query when any comparision operation is to be preformed with in the Query.

All

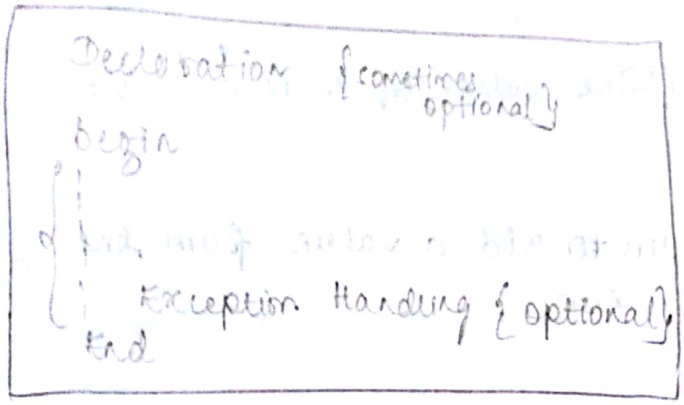
(Q) WAQ to display max rating of sailors

- select max (rating) from sailors s; → using aggregate function max
- Select s.sid from sailors where s.rating > all (select s.rating from sailors s); → without aggregate functions.

PL/SQL (procedural Language Structured Query Language)



Syntax -



6/9/2017

Write a PLSQL program to display a message "Hi friends".

predefined keywords

```
begin
dbms_output.put_line("Hi friends");
end;
```

→ This is like printf statement in C.

If we write in quotes as it is if i.e will be printed. If not in quotes, the value present in that variable will be printed.

- After each program we write back slash (/)

(Q) Write a PLSQL program to initialize a variable with value 10 and print the same.

```
declare
x number(5) := 10;
begin
dbms_output.put_line (x);
end;
```

(OR)

```
declare
 x  number(5):=10;
 begin
 dbms_output.put_line ('The value of x is :' || x);
 end;
/
```

Q) Write a PLSQL program to aid a value from the keyword and print the same.

(A)

```
declare
   X number (5);
   begin
    X = '&x'.
   dbms_output.put_line ('The value of x is : ' || x );
   end;
/
```

(Q) Write a PLSQL program to find sum of 2 numbers and display the same by reading the values through keyboard.

```
declare
x numbu(5);
y number(5);
z number(5);
begin
   x := '&x';
   y := '&y';
   z : x+y;
dbms_output.put_line ("The value of z is :' || z);
end;
```

(5) Write a PL/SQL program to print numbers from 1 to 10

```
declare
x number(5);
begin
for x in 1..10
 loop
dbms_output.put_line (x);
end loop;
end;
/
```

for printing 10 to 1 :

```
declare
x number(5);
 begin
  for x in reverse 10..1
  loop
 dbms_output.put_line (x);
 end loop;
 end;
/
```

Same program using while loop.

```
Declare
x number(5) := 1;
begin
while x<10
loop
dbms_output.put_line (x);
 x=x+1;
 endloop;
loop; end;
/
```

(OR) if we don't know boundries.

```
declare
x number(5);
begin
    x = &x;
    .
    .
    .
  end;
/
```

decrement

```
Declare
... = 10
.
.
while x<1
  .
  .
  .
end;
/
```

simple loop

```
declare
 x number(5):=1;
 begin
  loop
   dbms_output.put_line (x);
   i=i+1;
   exit i>10;
   end loop;
  end;
  /
```

Q) Write a PLSQL program to display the following content.

(i) if marks >= 75 'Distinction'
(ii) if marks < 75 'first class'.

```
declare
x number (10);
begin
 x='&x';
 if x>=75 then
 dbms_output.put_line ('Distinction');
else
 dbms_output.put_line ('First class');
 end if;
 end;
/
```

(Q) Write a PLSQL program to display the following

(i) if age = 30 'adult'
(ii) if age = 16. 'teenage'.
(iii) if age = 10 'child'
(iv) for all other condition display 'unknown'.

(4)

```
declare
x number(10);
begin
x := &x
if x := 30 then
dbms_output.put_line ('adult');
elsif x := 14
dbms_output.put_line ('teenage');
elsif x := 10
dbms_output.put_line ('child');
else
dbms_output.put_line ('unknown');
end if;
end;
/
```

x is 30.

- Here dbif spelling is elsif.
- If we write elsif, we need not put endif for every if.

Active Database -

If any modifications are made within the database by the user. that modifications are reported to DBA. such database will be called as active database.

In order to create active database we use concept of triggers.

- A Trigger

A Trigger is a procedure that will be automatically excecuted when any modifications are made on a database table.

working of trigger program -

The following -

Event - Any modification made on database table which invokes trigger program.

Condition - Condition written inside trigger program that performs action the result is true.

Action - Necessary action that is to be performed by the trigger once the condition is satisfied.

Types of trigger -

Row level (for each row)

Syntax

```
create or replace trigger <trigname> [before/after]
[insert/delete/update] on <table name>;
for each row
declare
=
=
begin
=
=
end;
```

Normal PL SQL

- In this Row level trigger a statement called 'for each row' will be present.
- Trigger will be invoked at each row, necessary action will be performed and it will be off at the end of the row.

(6) Write a trigger on Employee table to display message 'modifications made on employee table when any update operation is performed.

```
create or replace trigger trig_emp1
before update on employee;
for each row                    → condition
begin                           → action
dbms_output.put_line ('modifications made');
end;
/
```

In how many rows modifications are made that many times msg will be printed.

Statement level trigger

Syntax

```
create or replace trigger <trigname> [before/after]
[insert/delete/update] on <table name> ;
declare
=
begin
  =
end;
/
```

Same above ex

```
create or replace trigger trig-emp1
before update on employee;
begin
dbms_output.put-line ('modifications made');
end;
/
```

- In this statement level trigger a statement 'for each row' will not be present
- Trigger will be invoked only once and it will be on until the last record of the table is executed.

(Q) Write a trigger to display details of user who made modification on employee table.

```
create or replace trigger trig_emp1
before insert or update or delete on employee;
for each row;
declare

var varchar(10);
begin
select user into var from dual;
dbms_output.put_line (var || 'modified employeetable');
end;
/
```

(Q) Write a trigger program to display a message new record inserted into the table when any user inserts a new record on student table.

```
create or replace trigger trig_student before
insert on student;
for each row;
declare
var varchar(10);
begin
select user into var from dual;
dbms_output.put_line (var || 'new record inserted');
end;
/
```

Embedded SQL-

08.09.2017.

For an user to interact with database we need an intermediate programming language or interface called as past programming language SQL statements embedded within this past prog. lang will be called as embedded SQL.

- Drawbacks

In order to embed or insert into SQL statements we generally face two problems

(i) Type mismatch

Data type of SQL will not match with datatype of past programming language

(ii) Variables declared in past programming language cannot handle multiple columns or records at a particular point of time.

To solve these difficulties we use cursors

Cursors-

- It is a work bench/space where SQL statements will be excecuted.
- There are two types of cursors
  - implicit cursor.
  - Explicit cursor.

Implicit cursor -

- Implicit cursor is similar to general PL/SQL program and no additional key words are used here.
- This cursor is suitable to perform operations like delete, update and insert record within an existing table.
- It is not suitable to perform select operation.
- Attributes used in implicit cursor.
  - (i) sql %.found — T (condition block), F (Else block)
  - (ii) sql %.notfound - Reverse of found.
  - (iii) sql %.row count - gives row count of no of modifications.

Eg-

```
begin
if sql%found then
update employee set salary = 10000 where salary = 5000;
dbms_output.put_line (sql%rowcount || 'records got updated');
else
dbms_output.put_line ('records not found');
end;
/
```

(Here DB will know whether or not records or updated or not by attributes).

it can be delete, insert also

(OR)

```
begin
update employee set salary = 10000 where salary = 5000;
end;
/
```

Explicit Cursor

- Explicit cursors contains certain keywords that are to be followed.
- Explicit cursors are suitable for retriving the data and displaying it

Steps to create Explicit cursor -

1) declaration of cursor :-

Syntax- cursor <name_of_cursor> is select ....;

should not match with any other table name

2) Open Cursor -

Open <cursor_name>;

3) Fetch values from cursor to the table

fetch <cursor_name> into var1, var2, ...;

by this step values will be imported into table.

4) Close cursor -

Close <cursor_name>;

Q) Write a PLSQL procedure to display employee-id of all employees using the concept of cursors.

```
declare
var1 number(10);
cursor cur1 is select eid from employee;
begin
 open cur1;
 loop
  if cur1%.found then
fetch cur1 into var1;
dbms_output.put_line(var1 || cur1%.rowcount );
end
exit;
end if;
 end loop;
 close cur1;
end;
/
```

Attributes of Explicit Cursor-

- <cursor_name>%.found → T / F
- <cursor_name>%.notfound → T / F
- <cursor_name>%.rowcount - It tells the row count that is modified.
- <cursor_name>%.is open - We use. It tells whether the cursor is open or not.
  - T
  - F

Drawbacks of above program

- Depending on the number of columns that are to be displayed we declare equal number of variables.
- While declaring variables in PLSQL program the data type of variable should match with data type of columns stored in the table.

13/09/2017

UNIT-4

- NORMALIZATION OF DATABASE TABLES -

Normalization -

- It is the process of performing schema refinement by decomposing a single table/relation into two or more relations if necessary
- This normalization process will reduce redundancy present in the relations.

Problems caused due to redundant data/redundancy.

1. Insertion Anamoly.

In functional dependency FD :- $R \xrightarrow{\text{determines}} S$ if we are trying to insert dependent column without inserting values into determinant column is called as insertion anamoly

2. Updation Anamoly -

If we are trying to make updations in the dependent column with reference to some other column other than determinant then it is updation anamoly.

3. Deletion Anamoly -

It is not possible to delete determinant column within a table due to existing functional dependencies present in a table.

Functional Dependency - (FD)

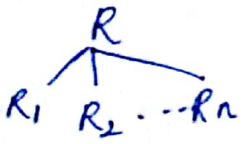
If R is a relation/table and X, Y are columns of existing within the relation then FD :- $X \rightarrow Y$ (X determines Y)

where X is determinant

Y is dependent.

Decomposition -

If 'R' is a relation, the process of dividing R into two or more smaller relations is called as decomposition



(i.e) $R = \{ R_1 \cup R_2 \cup \ldots R_n \}$

Closure Set - (Z^+)

- Set of all functional dependencies that can be determined by set 'S' will be called as closure of that set (Z^+)

Algorithm -

Armstrong's Axiom / Inference rules

1. Reflexivity - If b is subset of A ; $A \rightarrow B$
2. Augmentation :- If $A \rightarrow B$, then $AC \rightarrow BC$
3. Transitivity :- If $A \rightarrow B$ & $B \rightarrow C$ then $A \rightarrow C$

Other Rules -

Decomposition :- If $A \rightarrow BC$ then $A \rightarrow B$ and $A \rightarrow C$

Union :- If $A \rightarrow B$ and $A \rightarrow C$ then $A \rightarrow BC$

Composition :- If $A \rightarrow B$ and $C \rightarrow D$ then $AC \rightarrow BD$

Pseudotransitive rule :- $A \rightarrow B$ and $BC \rightarrow D$ then $AC \rightarrow D$.

Algorithm

Attributes of closure -

computing Z^+ of Z under "S"

closure [Z, S] = Z ;

do "forever"

for each FD $X \rightarrow Y$ is S

do

if $X \subseteq$ closure [Z,S]

then

closure [Z,S] = closure [Z,S] $\cup$ Y ;

end

if closure [Z,S] do not change on this iteration then

end loop;

end ;

/

Problems

(1) If R is a relation having attributes A, B, C, D, E, F and functional dependence is

$A \to BC$,
$E \to CF$,
$B \to E$,
$CD \to EF$. Compute closure of $S = \{A, B\}$.

Solution-

	F.D		Closure
ITERATION-1	$A \to BC$	$S = \{A, B\}$ $A \subseteq \{A, B\}$ $\therefore BC \cup \{A, B\} = \{A, B, C\}$	$\{A, B, C\}$
	$E \to CF$	$E \not\subseteq \{A, B, C\}$	$\{A, B, C\}$
	$B \to E$	$B \subseteq \{A, B, C\}$ $E \cup \{A, B, C\}$	$\{A, B, C, E\}$
	$CD \to EF$	$CD \not\subseteq \{A\ B\ C\ E\}$ → if any one is not in the set it is not in the S. Hence closure remains same.	$\{A, B, C, E\}$
Iteration-2	$A \to BC$	$S = \{A, B, C, E\}$ $A \subseteq \{A, B, C, E\}$ $\therefore BC \subseteq \{A, B, C, E\}$	$\{A, B, C, E\}$
	$E \to CF$	$E \subseteq \{A, B, C, E\}$ $\therefore CF \cup \{A, B, C, E\}$	$\{A, B, C, E, F\}$
	$B \to E$	$B \subseteq \{A, B, C, E, F\}$ $\therefore E \cup \{A, B, C, E, F\}$	$\{A, B, C, E, F\}$
	$CD \to EF$	$CD \not\subseteq \{A\ B\ C\ D\ E\ F\}$	$\{A, B, C, E, F\}$

FD		Closure
	$S = \{A, B, C, E, F\}$	$\{A, B, C, \bullet, E, F\}$
$A \to BC$	$A \subseteq \{A, B, C, E F\}$	
		$\{A, B, C, E, F\}$
$E \to CF$	$E \subseteq \{A, B, C, E F\}$	
		$\{A, B, C, \cancel{E F}\}$
$B \to E$	$B \subseteq \{A, B, C, E, F\}$	
		$\{A, B, C, E F\}$
$CD \to EF$	$CD \not\subseteq \{A, B C E F\}$	

Iteration-2

Result of II = Result of III iteration

$$Z^+ = S^+ = \{A, B, C, E, F\}$$

(2) If R is a relation, A, B, C, D, E, F, G are the attributes and functional dependences $A \to B$; $BC \to DE$; $AEF \to G$.
Find closure of $S = \{A, C\}$

solution –

	Functional dependencies		closure
Iteration I	$A \to B$	$S = \{A, C\}$ $A \subseteq \{A, C\}$ $\therefore B \cup \{A, C\}$	$\{A, B, C\}$
	$BC \to DE$	$BC \subseteq \{A, B, C\}$ $DE \cup \{A, B, C\}$	$\{A, B, C, D, E\}$
	$AEF \to G$	$AEF \not\subseteq \{A, B, C, D, E\}$	$\{A, B, C, D, E\}$
Iteration II	$A \to B$	$S = \{A, B, C, D, E\}$ $A \subseteq \{A, B, C, D, E\}$	$\{A, B, C, D, E\}$
	$BC \to DE$	$BC \subseteq \{A, B, C, D, E\}$	$\{A, B, C, D, E\}$
	$AEF \to G$	$AEF \not\subseteq \{A, B, C, D, E\}$	$\{A, B, C, D, E\}$

Result of I = Result of II

$\therefore Z^+ = S^+ = \{A, B, C, D, E\}$

MODEL-II

If R is a Relation having attributes A, B, C, D, E, F where functional dependencies $A \to BC$, $B \to E$, $CD \to EF$

Show that F.D $AD \to F$ holds for R

Solution :-

$A \to BC$

$A \to B$, $A \to C$ {By decomposition}

$A \to C$ $\quad AD \to CD$ {Augmentation Rule}

$AD \to CD$, $CD \to EF$ {Transitivity}

$AD \to EF \Rightarrow AD \to E$ & $AD \to F$ {Decomposition}

Types of Functional Dependences -

1) Full FD -

If x, y or attributes of relation ~~are~~ R then $x \to y$ is said to be fully functionally dependent only if $x \to y$ and no other subset of x is ~~also~~ able to determine y.

$x \to y$
and $p \subseteq x\ (p \not\to y)$

Ex - $AD \to BC$ [Full FD]
$A \to BC$ [A→BC is in Full F.D or NO]
$B \to E$

$AD \to BC$ is fully functionally dependent as BC is dependent on AD and no other subset can determine BC.

2) Partial FD -

If x determines y then it is said to be partial F.D only when an element p (is a subset of x) will also determines y (ie) $\exists p \subseteq x; p \to y$

Ex - $AD \to BC$ [R]
$A \to BC$ [A→BC is in Fully F.D]
$C \to D$

Sol - $AD \to BC$
$A \to BC$

But $A \subseteq AD$
Therefore, $A \to BC$ is partial F.D]

3) Transitive F.D -

If x determines y and y determines z then x determines z

$X \to Y$; $Y \to Z$ then $X \to Z$

Ex - T:- $AD \to BC$
FD_2 { $A \to BC$
FD_3 { $BC \to D$

Sol :- $A \to BC$
$BC \to D$
$A \to D$.

4) Trivial FD -

If $X \to Y$ But $Y \subseteq X$

eg- $ABC \to BC$; $BC \subseteq ABC$ is trival F.D.

100% sem Q.

* Normal Forms / Normalization procedure -

→ 1st Normal Form - 1NF

A relation is said to be in 1NF when all its attributes or single values (composite columns are not permitted).

Example -

Student

Rollno	name	subjects
1	Priya	{DBMS, SE, DAA, FLAT, ...}
2.	Nikitha	"
3.	Teja	"

→ multivalued attributes are not to be considered.

The following are methods

Method1

ROLLNO	name	subject
1	priya	DBMS
1	priya	SE
1	priya	DAA
1	priya	FLAT
:	:	:
2	Nikitha	DBMS
2	Nikitha	SE
:		

Method2 decomposition

RollNo	Name
1	Priya
2	Nikitha
3	Teja

Rollno	subject
1	~~Priy~~DBMS
1	DAA
1	SE
:	:
2	DBMS
2	DAA

2nd Normal Form 2NF

A relation is said to be in 2NF if it satisfies the following properties

(i) It should be in 1NF

(ii) All the non-prime attributes should be fully functionally dependent on prime attribute.

(iii) Transitive dependency is also permitted.

Note -

- prime attribute - key attributes of a relation
- Non prime attributes - Non-key attributes of a relation.

Example- Consider R is a relation having attributes ABCDEF and FD's

FD1 : $A \rightarrow BCDEF$

FD2 : $BC \rightarrow ADEF$

FD3 : $B \rightarrow F$

FD4 : $D \rightarrow E$

Step1 - Identify key attributes for the above functional dependencies.

key - $\{A\}^+ = \{A, B, C, D, E, F\}$

$\{BC\}^+ = \{B, C, A, D, E, F\}$.

'A and 'BC are key Attributes for the above given FD's.

another ex-

$A \rightarrow BCE$

$D \rightarrow B$

$E \rightarrow F$

$\{ADE\} \rightarrow \{A, B, C, D, E, F\}$

Step2

Prime- ABC

Non prime- DEF

FD1:- $A \rightarrow BC\boxed{DEF}$

decompose FD1

$A \rightarrow BC$; $A \rightarrow DEF$ {Fully functionally dependent}

FD2 - $BC \rightarrow ADEF$ {Not fully F.D}

as in every FD, BC⊆BC is $B \rightarrow F$

FD3 :- B → F

FD1 :- A → CDEF ; A → B (decomposition)

from FD1 and FD3

A → B ; B → F

∴ A → F (Transitive FD)

FD4 :- D → E

It is not FD on prime attribute

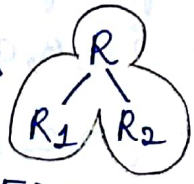
from FD1 :- A → BCEF, A → D [decomposition]

Therefore, from FD1 and FD4

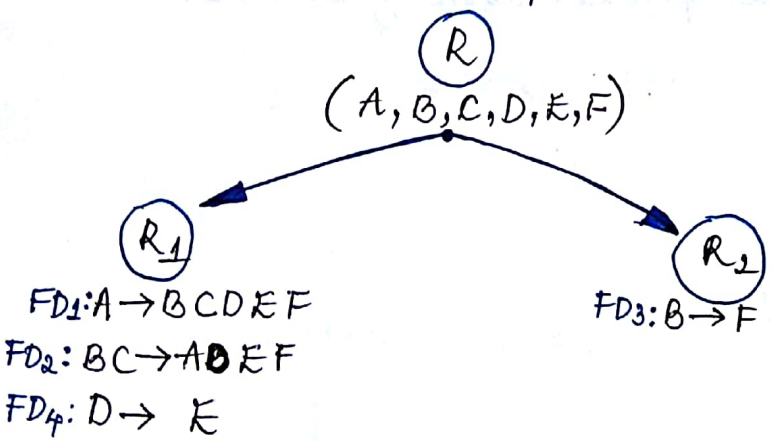
A → D ; D → E

Hence, A → E (Transitive dependency].

Step3- From step 2 we can identify that FD2 is not fully functionally dependent on prime attribute because of FD3 so, we decompose relation R into two parts such that



R1 holds FD1, FD2, FD4 and R2 holds FD2.



3rd Normal Form (3NF)

A relation R is said to be in 3NF if it satisfies the following terms
(i) It should already in 2NF.
(ii) All the non prime attributes should be ~~fully~~ fully F.D on prime attribute.
(iii) Transitive dependency is not permitted.

problem continution,

R_1: $A \rightarrow BCDEF$
$BC \rightarrow ADEF$
$D \rightarrow E$

prime - ABC
Non prime - DEF

FD1 :- $A \rightarrow BCDEF$
decompose FD1
$A \rightarrow BC$; $A \rightarrow DEF$ (fully F.D)

FD2 :- $BC \rightarrow ADEF$ (fully F.D)
As in FD3 $BC \subseteq BC$ is $B \rightarrow F$

FD4 :- $D \rightarrow E$
from FD1 :- $A \rightarrow BCEF$, $A \rightarrow D$ (decomposition)
Therefore from FD1 and FD4
$A \rightarrow D$; $D \rightarrow E$ $\therefore A \rightarrow E$ (Transitive dependency.
As transitive dependency is not permitted

(R)
(A, B, C, D, E, F)

(R1) (R2)
$B \rightarrow F$

(R3) (R4)
$D \rightarrow E$

$A \rightarrow BCDEF$
$BC \rightarrow ADEF$

BCNF - Boyce Codd Normal Form

A relation R is in BCNF so, it should follow the terms below

(i) It should be 3NF

(ii) All the non-prime attributes should be dependent on super key only

R_2: $\boxed{B} \rightarrow F$ - Key attribute is B

R_3: $\boxed{A} \rightarrow BCDEF$
$\boxed{BC} \rightarrow ADEF$
] Key attributes A, B, C

R_4: $\boxed{D} \rightarrow E$ — Key attribute is D

In BCNF we are here to find out key attributes for all relations.

Therefore, the given relation is converted into BCNF.

— Solved —

(2.) $AB \rightarrow CDE$

$B \rightarrow C$

$D \rightarrow C$

Convert into 1NF, 2NF, 3NF and BCNF.

Higher Normal form -

4th Normal form -

A relation is said to be in 4NF if it satisfies the following terms.

(i) It is in BCNF

(ii) A relation should not contain multi-valued dependencies.

Employee :

eid	lid	name	gid	gname
501	1	T	101	chess
	2	H	102	chess
	3	E	103	chess
502	1	T	101	tennis
	2	H	102	tennis
	3	E	102	tennis

from the above table two 1:N dependencies are present But according to 4NF, multiple dependencies should not exist within the same relation R. Hence, we decompose the above relation into two relation R_1 and R_2.

R_1

Emp_lang

eid	lang_id	lname
501	1	T
501	2	H
501	3	E

R_2

emp_sports

eid	gid	gname
501	101	chess
501	102	tennis
501	103	cricket

5th Normal Form -

A relation is said to be in 5NF if it satisfies the following
(i) It should be already be in 4NF
(ii) Lossey not be done.
(The decomposed tables are to be combined into the original table after decomposed table is equal to table before decomp.)

Note - If $R_1 \bowtie R_2 \bowtie \ldots R_n = R$, then it is in 5NF

natural join

eid	lid	lname	fid	fname
501	1	T	101	chess
501	1	T	102	tennis
501	1	T	103	carroom
501	2	H	101	
501	2	H	102	
501	2	H	103	
501				

properties of decomposition.
*BM (i) loss less join - decomposed to original
(ii) Preserving functional dependencies
(ii) loss less join
A decomposition $D = \{R_1, R_2 \ldots R_n\}$ of R.

GITAM University, Hyderabad
School of Technology
Mid Term Examinations-II

Branch/ Semester: CSE/ V
Subject Code: EID301
Subject Name: DBMS

Date:29.08.2017
Duration: 75 Minutes
Max.Marks: 30

(Answer to Q1 is Compulsory) **(5x2=10 Marks)**

Q. 1
a) How to retrieve records from a relation by eliminating duplicate records.
b) What is update restriction on views?
c) What is the representation format for TRC and DRC?
d) Distinguish between UNION and UNION ALL.
e) What is enforcing integrity constraints on database.

(Answer any four questions from the following) **(4x5=20 Marks)**

Q.2.What is querying relational data from the database? Explain.

Q. 3. Write a short note on following terms:
(a) Division operation
(b) Except
(c)Rename

Q.4.What is a view table? Differentiate between simple and complex view tables.

Q.5. Consider the following Relation Schema
Sailors(*sid* number, *name*:string, *age*:number)
Reserves(*sid* :number, *bid*:number,*day*:date)
(a)Write a query to retrieve names of sailors who have reserved boat with bid=103
(b)Write a query to retrieve details of sailors who have reserved atleast one boat.

Q.6. Consider the following Relation Schema, Write the following queries in relational algebra.
Suppliers(*sid:* integer, *sname:* string, *address:* string)
Parts(*pid:* integer, *pname:* string, *color:* string)
Catalog(*sid:* integer, *pid:* integer, *cost:* real)
(a) Find the *names* of suppliers who supply some red part.
(b)Find the *sids* of suppliers who supply some red or green part.
(c)Find the *sids* of suppliers who supply some red part or are at 221 Packer Street.
(d)Find the *sids* of suppliers who supply some red part and some green part(set operators)
(e) Find the Supplier names of the suppliers who supply a green part that costs less than 100 dollars.

GITAM University, Hyderabad Campus
School of Technology
MID Term Examinations-I

Branch/ Semester: CSE/V
SubjectCode: EID301
Subject Name: DBMS

Date:21/07/2017
Duration: 75 Minutes
Max.Marks: 30

(Answer to Q1 is Compulsory)

Q. 1 (5x2=10 Marks)

a) Differentiate between conceptual data independence and physical data independence.

b) Specify two kinds of constraints with respect to ISA hierarchies.

c) What is the difference between UNIQUE key and Primary Key in relational Model?

d) Discuss the use of descriptive attribute with an ER-Diagram

e) What is the degree of a relation in the Relational Model?

(Answer any four questions from the following) **(4x5=20 Marks)**

Q. 2 . Why would you choose a database system instead of file system? Discuss the advantages of DBMS.

Q. 3 Discuss about (i) an Entity (ii) an Attribute (iii) Relationship and (iv) Weak Entity.
Convert below data into ER-diagram:

(a)Employee (eid: integer, ename: string, age: integer, salary: real, DOJ: date, phone_numb: integer). eid as primary key.Phone_numb as multivalued attribute.
(b) Dependent (eid: integer, dname: string, age: integer). Eid is partial key.

Q. 4 Distinguish between partial participation and total participation with an ER-diagram

Q. 5 Discuss the different Integrity Constraints over Relations in the Relational Model?

Q. 6 What are the Different types of Data Models? Explain about Relational Model?