S. No.	Date	Title	Page No.	Teacher's Sign / Remarks

"Data Base Management System"

7/10/24

- Data :- The fact that can be recorded is called as Data.
- Database :- Collection of related data is known as data base.

properties of Database :-

i] A Database represents some aspects of real world, sometimes called as miniworld or universe of discourse

ii] A Database is logically coherent collection of data with some inherent meaning

iii] A Database is designed, build and populated with data for a specific purpose (It has an intended group of users)

iv] Database can be of any size and complexity.

v] A Database may be generated and maintained manually or it may be computerized.

What is DBMS:

A DBMS is a collection of programs that enables users to create and maintain a database.

In other words DBMS is a general purpose software that facilitates the process of defining constructing manipulating and sharing databases among various users and applications.

* Defining the database :-

It involves specifing the datatypes, structures and constraints of the data to be stored in the database

* Constructing the database :-

Constructing the database is a process of storing the data on some storage media that is controlled by DBMS

* Manipulating database :-

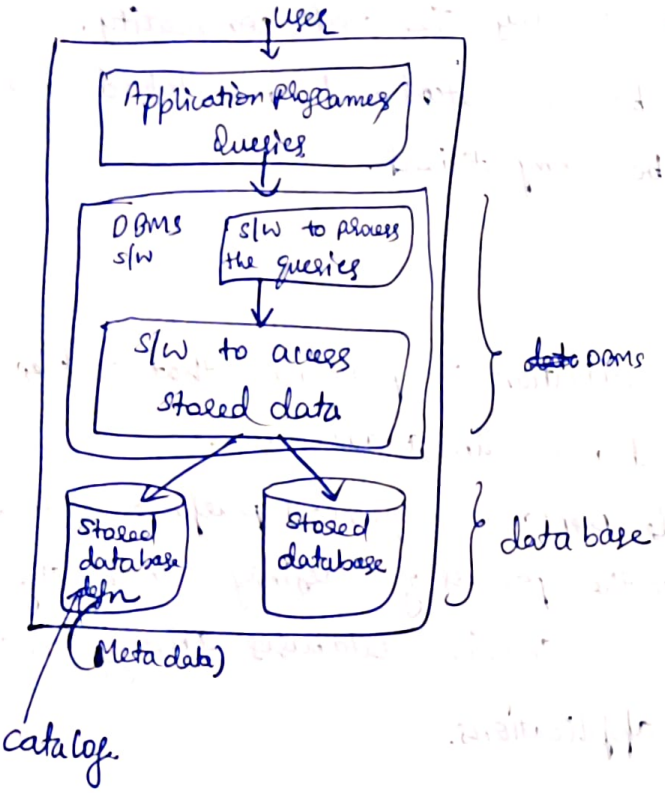
It includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the data, and generating reports from the database.

* Sharing the data :-

Sharing a database allows multiple users and programs to access the database simultaneously

A simplified database system Environment :-

database system = database + DBMS software



"Database System"

- protecting the database:- This includes system protection against Hardware or software failure and security protection against unauthorized access

- Maintanance:-
  This includes maintanance of the database system by allowing the system to evolve as requirements changed over time

9|10|24

Functionalities of DBMS:-

- Defining the database
- constructing the database
- Manipulating the database
- Sharing the database
- protecting database
- Maintenance.

Characteristics of database approach:-

[Difference between the traditional file processing approach and Data base approach]

1] Self describing nature of DBMS:

- In DBMS approach database system contains not only the database itself but also a complete defination or description of database called as Metadata and is stored in catalog

- In traditional file processing data definition is typically part of application program, hence these programs work with only one specific database. Where as DBMS is a general purpose software which is not written for a specific database application and can access diverse databases by extracting the database definition from the catalog and then using these definations.

2] Insulation between programs and data and data abstraction

- The structure of data files is stored in the DBMS catalog separately from the access program this property is called as program data independence.

- In some type of database systems such as object oriented and object relational the implementation of operation on the data is specified separately and can be changed without affecting interface, uses application program can operate on data by invoking these operations through their name and arguments regardless of how operations are implemented this property is called as "program operation independence"

data abstraction = program data independence + program operation independence

3] Support of multiple views of data

- The database typically has many users each require a different perspective or view of database, A view is a subset of database or it may contain virtual data that is derived from the database files but it is not explicitly stored.

4] Sharing of data and multiuser transaction processing

- To handle multiple users DBMS includes concurrency control software to ensure that several users trying to update same data at the same time, in a controlled manner so that the result of the updates is correct.

- This type of applications where multiple users trying to access the same data at the same time are called as "OLTP" applications.
  ↳ Online transaction processing applications

- Here DBMS concurrency control mechanism ensures that concurrent transactions operate correctly and efficiently.

- Database approach also provides the concept of transaction which is an executing program or process that includes one or more database access such as reading or updating the database records. The DBMS must enforce several transaction properties such as atomicity, isolation and so on.

⇒ Data Models :

A data model is a collection of concepts that can be used to describe the structure of a database and provides the necessary means to achieve the abstraction

Catogories of data models :

i] **High level or conceptual data model :**

It provides the concepts that are closed to the way many users percieve the data

Ex:- ER diagram

ii] **Low level or physical data model :**

It provides concepts that describes the details of how data is stored in the computer

iii] **Representational or Implementation data model :**

Which provides concepts that may be understood by some end users but also includes some concepts that provides ~~three~~ way data is organized in a computer. These models can be implented on computer systems ~~directly~~ directly.

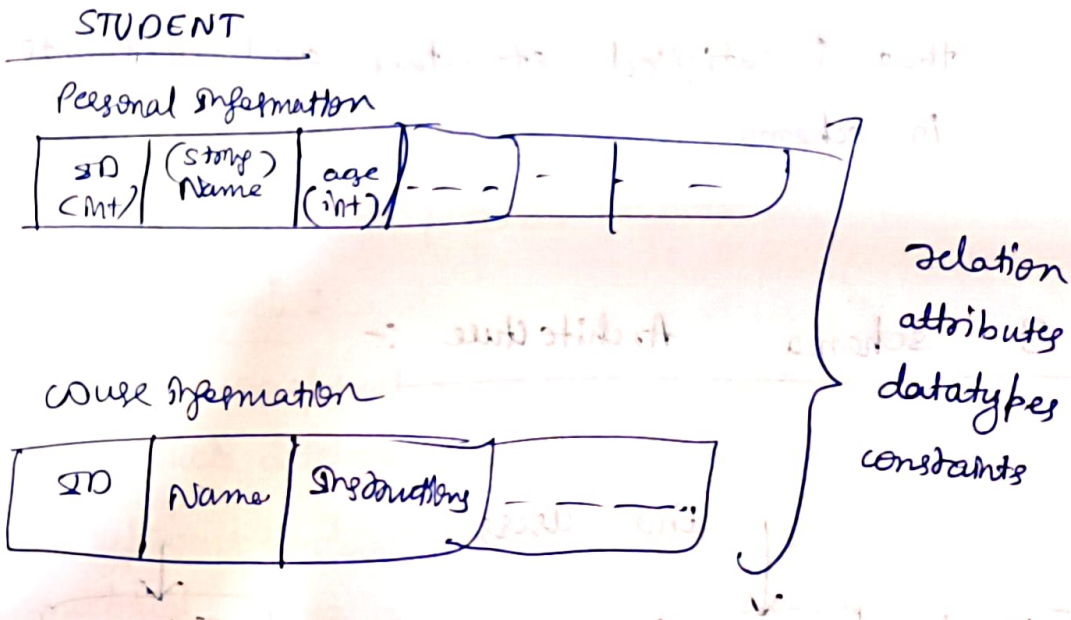
Ex:- Relational ~~DBMS~~ model

⇒ Schema :

The description of database is called as database schema, which is specified during database design and not expected to change frequently

→ Different data models have different ways of representing schema using diagram called as schema diagram

schema diagram for relational database:

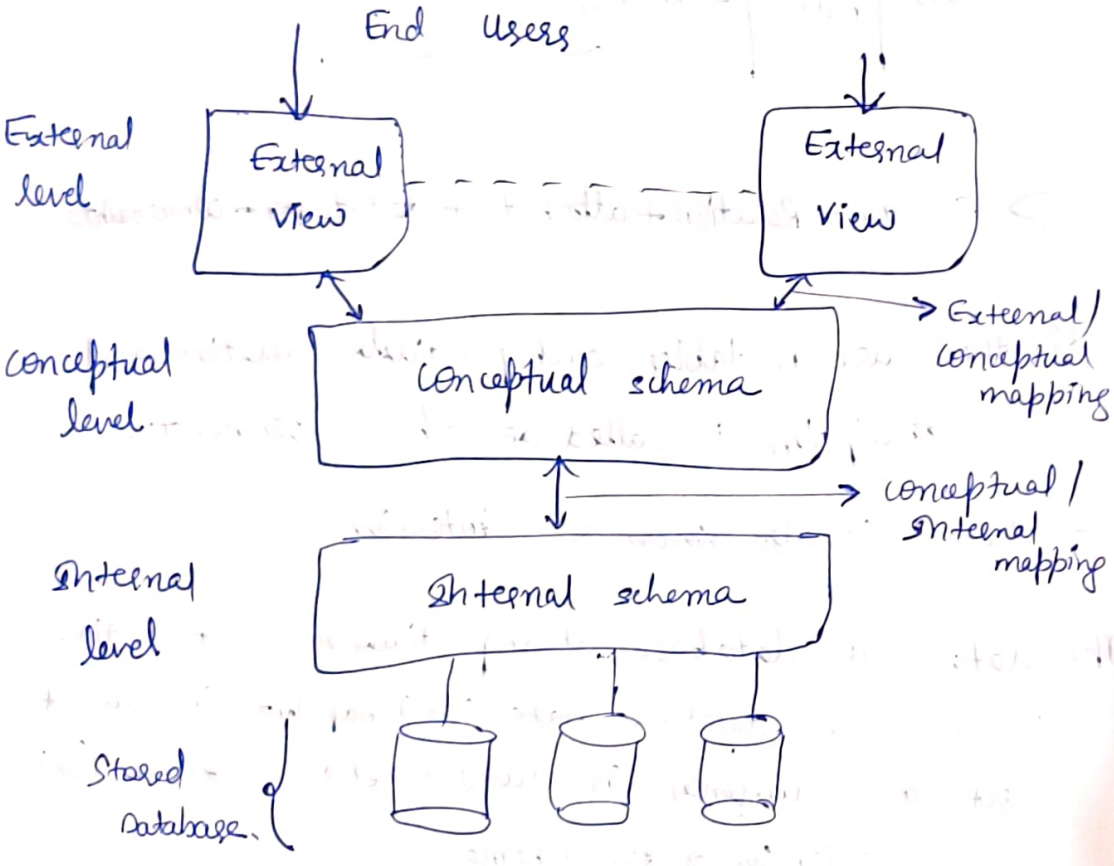


⟹ schema = Relation + attributes + datatypes + constraints

- If there are n tables each particulas relation and description is called as schema construct.
- schema is also known as intension
- The data in the database at a particular moment in time is called a database state (or) snapshot (or) current set of occurrences (or) current set of instances (or) extension of the schema
- When we define a new database we specify its database schema only to the DBMS, at this point the corresponding data base state is said to be empty state with no data. The database state is said to be initial state when the database is first populated or loaded with initial data from then on

everytime an update operation is apply to the database we get another database state. At any point in time database has a current state. The DBMS is partly responsible for ensuring that every state of database is a valid state. that is the state that is satisfied structure and constraints specified in schema

3 schema Architecture :-



10/10/24

The goal of 3 schema architecture is to separate user applications and the physical database. In this architecture schema can defined as the following 3 levels:

i) Internal level :-

This has an internal schema which describes the physical storage structure of the database. This uses a physical data model and describes a complete details of the data storage and access paths for the database.

ii) Conceptual level :

It has conceptual schema which describes the structure of a whole database for a community of users. Usually the representational data model is used to describe the conceptual schema when a a database system is implemented, this is based on a conceptual schema design in a high level data model

iii) The external or view level :

Includes a number of external schemas or user views each external schema describes the part of the database that a particular user group is interested in and hides a rest of the database from that user group.

Each external schema is typically implemented using a representational data model based on an external schema design in a high level data model.

The process of transforming requests and results between the levels is called as mapping accordingly we have external /conceptual mapping and conceptual/internal mapping.

**

Data model	level	Schema
High	External	External
Representational	Conceptual	Conceptual
Low	Internal	Internal

** Data Independence:-

3 schema architecture supports data independence which can be defined as the capacity to change the schema at one level of database system without having to change the schema at the next higher level, accordingly we have two types of data independence

i) Logical data independence:

It is the capacity to change the conceptual schema without having to change external schema of application program.

ii) Physical data independence:

It is the capacity to change the internal schema without having to change the conceptual schema. Hence external schema also need not be ~~same~~ changed.

DBMS Languages:

- In many DBMS's where no strict separation of levels is maintained the single language called as data defination language (DDL) is used by the DBA and by database designers to define ~~both~~ conceptual and internal schema's and external schema's (all schema's)
- In DBMS where a clear separation is maintained between the conceptual and internal levels. DDL is used to specify the conceptual schema only, another language called storage defination language (SDL) is used to specify the internal schema.
- In DBMS where a clear separation is maintained between the external conceptual and internal schema a new language called VDL stands for view defination language is used to specify external schema
- Once the database schema's are created and database is populated with data to perform manipulations like retrieval, insertion, deletion and manipution of the data is performed using a language called as Data manipulation language. (DML)
- There 2 types of DML:
  i) High level or non procedural DML
  ii) Low level or procedural DML

→ High level / procedural DML :

* These DML's specify what data is required from the database and does not specify the procedure to get the data. These DML statements can be entered interactively from a terminal or to be embedded in a general purpose programming language.

* This DML can specify and retrieve many records in a single DML statement therefore they are called as set-at-a-time (or) set-oriented DML's

→ Low level / Procedural DML :

* These DML's specify how to retrieve the data along with which data to retrieve, Hence they are also called as declarative

* These DML's typically retrieve individual records or objects from the database and process each record separately, hence they are called record-at-a-time DML's, Therefore to retrieve and process multiple records these DML's must be embedded in a general purpose programming language to use language constructs such as looping.

* Whenever DML commands statements are embedded in a general purpose programming language that language is called Host language and DML is called Data sub language

* A High level DML used in a stand alone interactive manner is called Query language.

* SQL is nonprocedural DML

* DDL, VDL, DML are supported by SQL.

*

DDL	DML
CREATE	INSERT
ALTER	DELETE
DROP	UPDATE

manipulating schema, table, key word

manipulating records

VDL

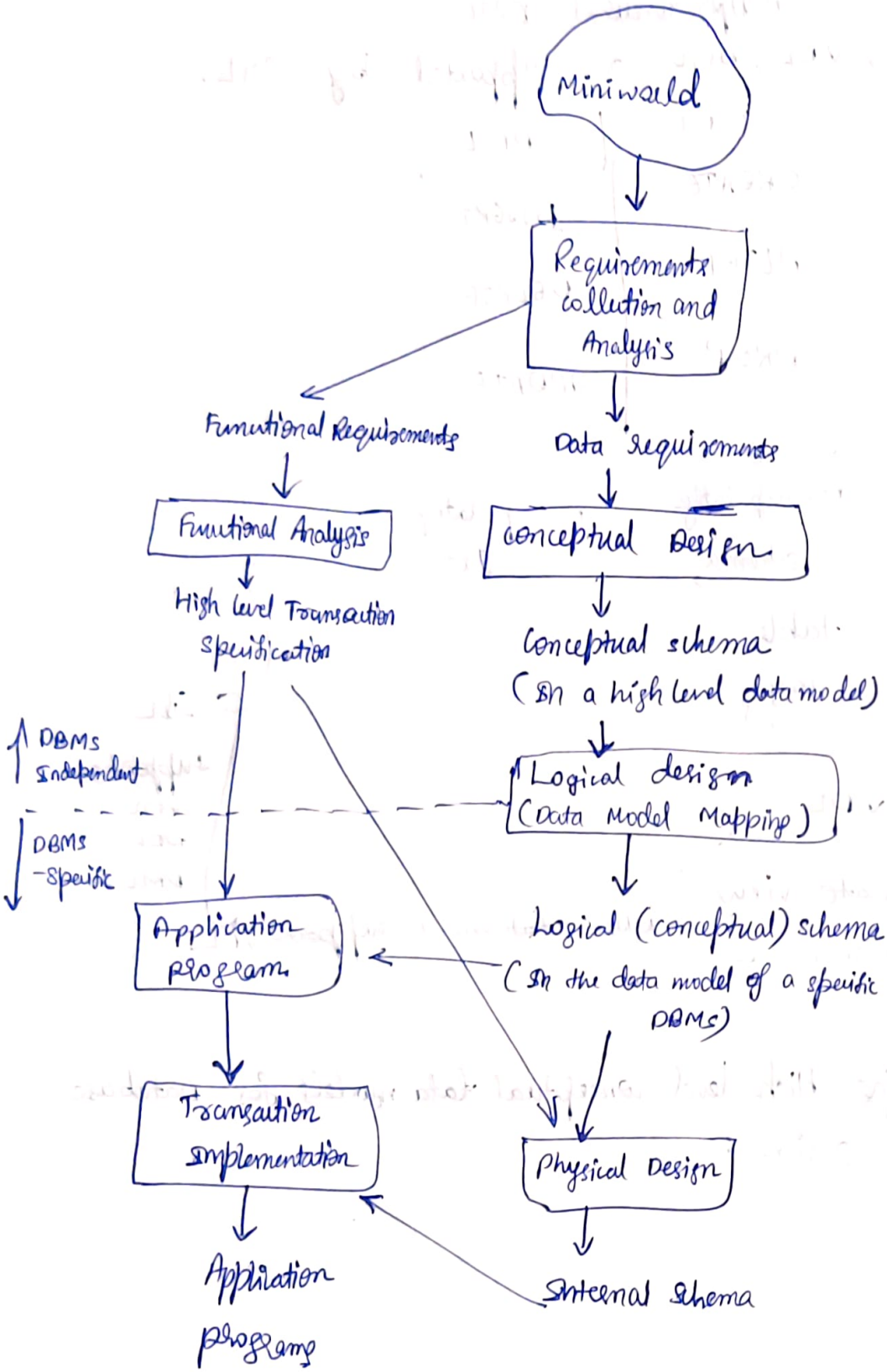
create view viewname } DDL statement supports VDL

SQL supports VDL, DDL, DML

⇒ Using High level conceptual data models for Database design :

PTO

A simplified diagram to illustrate the main phases of Database design



18/10/24

An example database application :-

A company database keeps track of a company's employees, departments and projects. Suppose that after the requirements collection and analysis phase a database designers provide the following description of the miniworld that is the part of company to be represented in the database.

i] The company is organize into departments, each department has unique name, a unique number and a particular employee who manages the department be keep track of the start date when that employee began managing the department, a department may have several locations.

ii] A department controls a number of projects, each project has unique name, a unique number and a single location

iii] We store each employee's name, social security number, address, salary, gender and birthdate. an employee is assigned to one department but may work on several projects which are not necessarily controlled by the same department, we keep track of the number of hours per week that an employee work for an each project; we also keep track of the direct supervises of each employee.

iv] We want to keep track of the dependents of each employee for insurance purpose, we keep each dependents first name, gender, birth date and relationship to the employee.

Entity :- The thing in the real world with an independent existence. It is an object with physical existence (person, city, house, so on ---) (OR) It's an object with conceptual existence (job, course ----)

Attribute :- Each entity has attributes where the attributes are particular properties that describe an entity

For ex :- EMPLOYEE is an entity

ssn, fn, mn, ln, Salary, address } Attributes of EMPLOYEE

Categories of Attributes in ER Model :-

1] Simple (Atomic) vs composite attributes :

→ Attributes that can be devided into smaller sub-parts which represent more basic attributes with independent meanings are called as composite attributes

For ex :- Address — country, Pincode } is considered as composite attribute.
Street, city

→ An attribute that cannot be devisable into subparts is called as simple or atomic attributes.

Ex :- Age

2] Single valued Vs multivalued :

→ An attribute that has a single value for a particular entity is called as single valued.

Ex:- Age

→ An attribute that takes more than one value for a particular entity is called as multivalued attribute

Ex:- phone numbers, degree of person

3] Stored vs Derived :

→ Any attribute whose values are explicitly stored in the database are called as stored attributes

Ex:- Birth date

→ An attribute whose value can be derived from a stored attribute is called as derived attribute

Ex:- Age

4] NULL values :

→ In some cases a particular entity may take null value for an attribute. The null is interpreted in following 3 different ways

i] Value is not applicable for the attribute.

ii] The value for the attribute is exist but it is missing.

iii] The value for attribute is not known.

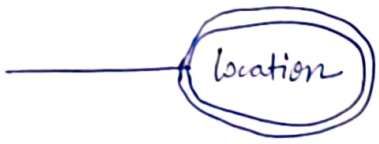
5] complex attributes :

→ These attributes are both composite and multivalued.

This is represented as follows

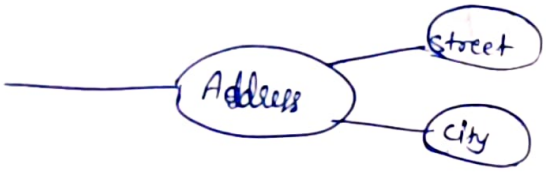
{ Address (street, city ----) }

Symbols for attributes representation



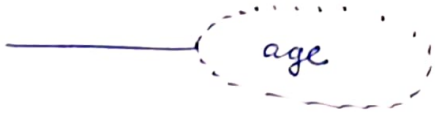
} multivalued

} simple



} composite

} complex



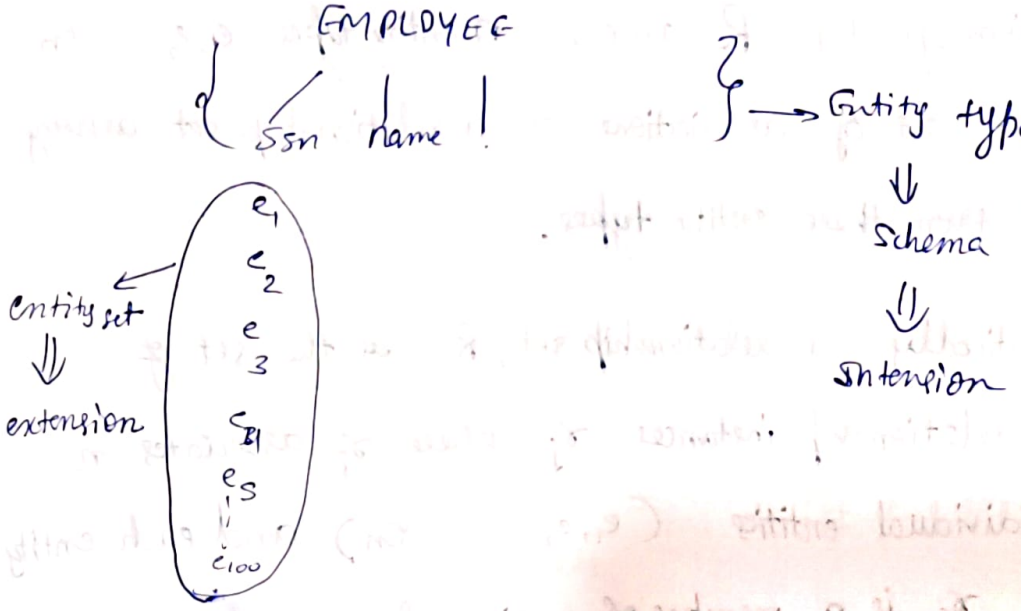
} Derived

Entity types :-

→ An entity type is a collection of entity that have the same attributes each entity type in the database is described by its name and attributes.

→ The collection of all entities of a particular entity type in the database at any point in time is called as entity set

Ex:



- In ER model entity type is represented by using rectangle.

Key attribute :-

The key constraint (or) uniqness constraints is an important constraint on the entities of an entity type.

An entity type has an attribute whose values are distict for each individual entity in the entity set, such an attribute is called as key attribute

The values of key attributes can be used to identify each entity uniqly, in ER diagram key attributes represented using regulas underline.

Values sets (Domains) Attributes :- [Domain is a datatype]

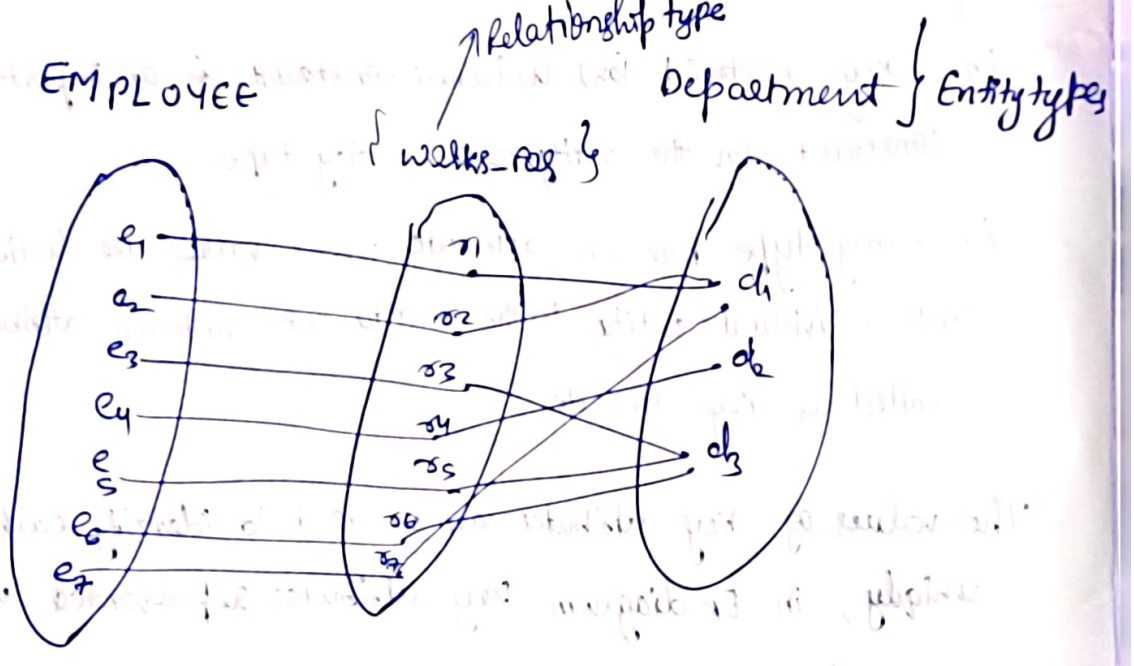
Values sets specifies set of values an entity can take in an entity type, usually value sets are defined using domain of the attribute

Relationship Types :-

- A relationship type R among n entity types $e_1, e_2 - - - e_n$ defines a set of associations or a relationship set among entities from these entity types.

- Mathematically a relationship set, R is the set of relationship instances r_i where r_i associates n idividual entities $(e_1, e_2 - - - e_n)$ and each entity e_i in r_i is a member of entity type e_i where $1 \leq i \leq n$

- Alternatively a relationship type $R \subseteq E_1 \times E_2 \times - - - - E_n$



$r_i \longrightarrow$ relationship instances

e_i and d_i } are entities

$$R = \{ r_1, r_2 - - - r_m \}$$

$$r_i = (e_i, e_j)$$

$$e_i \in E_i \text{ and } e_j \in E_j$$

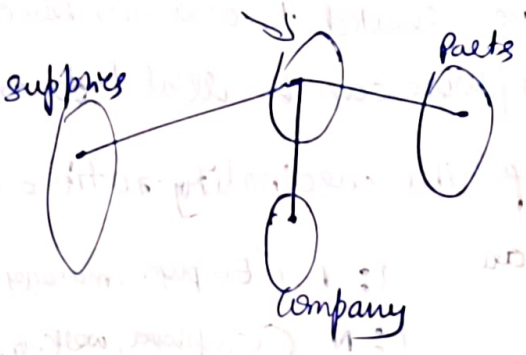
- Degree of a relationship type is the number of participating entity type.

- A relationship type of degree 2 is called binary,

For example: WORKS_FOR is binary relation type.

- If degree of relationship type is 3 then the relationship type is called ternary relationship type

Ex: SUPPLIES is ternary relationship type.

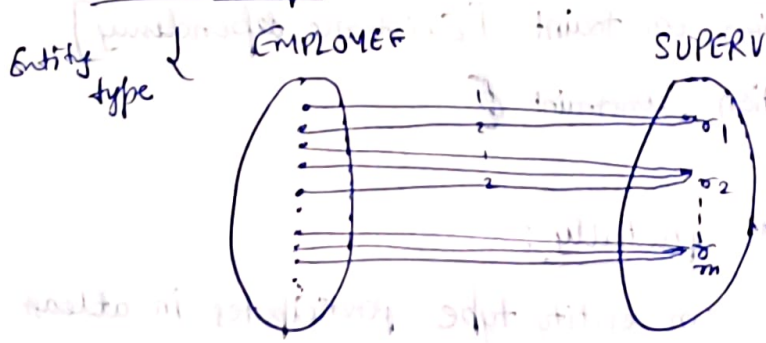


$r_i = (e_i, e_j, e_k)$

- Recursive relationship type:

If an entity type participates more than once in a relationship type we call the relationship type as recursive relationship type, in such relationship types specifying the role of the participating entity type is important.

For example:-



NOTE: Relationship type is represented in ER diagram using Rombhous

Constraints on Relationship type:

There are two constraints on relationship type

i) cardinality ratio for Binary relationship type:

It specifies maximum number of relationship instances that entity type can participate in.

For ex: In the WORKS_FOR binary relationship type DEPT: EMPLOYEE is 1:N indicates that each department can be related to any number of employees but an employee can be related to only one department. The possible cardinality ratio's for binary relationship types are

1:1 (Employee, manages, Dept)

1:N (Employee, works on, Dept)

M:N (Employee, WORK on, Project)

ii) Participation constraints: [minimum cardinality constraint]

This specifies the minimum number of relationship instances that each entity can participate in, hence it is also called as minimum cardinality constraint

two different types:

i) Total participation constraint [Existence dependency]

ii) Partial participation constraint

i) Total participation and partially:

If each entity of an entity type participates in atleast one relationship instance then we say that the participation of the entity type is total with respect to the relationship type otherwise we say that the participation of entity type with respect to relationtype is partially.

Note:- In ER diagram total participation is represented using double lines connecting entity type to relationship type.

* partially participation denoted using single line connecting entity type to relationship type.

Cardinality constraint

→ ~~total participation~~ + ~~partial~~ participation = structural constraint (min, max)

Constraint

- In ER diagram we use min, max representation to represent the structural constraint

- Weak entity types:

- Entity types that do not have key attributes of their own are called as weak entity types.
- The regular entity types that have key attribute are called strong entity types.
- Entities belonge to weak entity type are identified by being related specific entities from another entity type, in combination with one of their attribute values, we call this another entity type as identifying a owner entity type.
- The relationship type that relates a weak entity type to its owner is called as identifying relationship type
- In ER diagram identifying relationship types are represented by and weak entity types are represented by

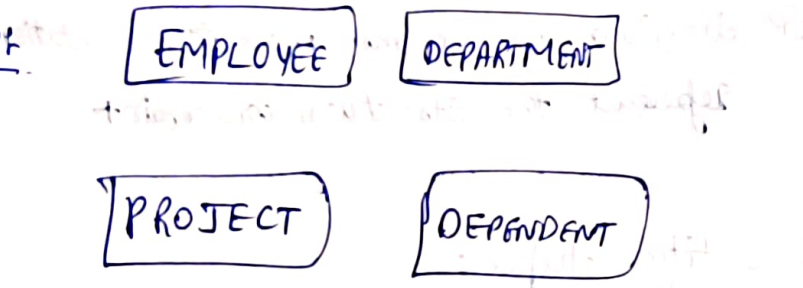
- A weak entity type normally has a partial key which is the set of attributes that can uniquly identify weak entities that are related to the same owner entity.

- In ER diagram partial key is represented by -----

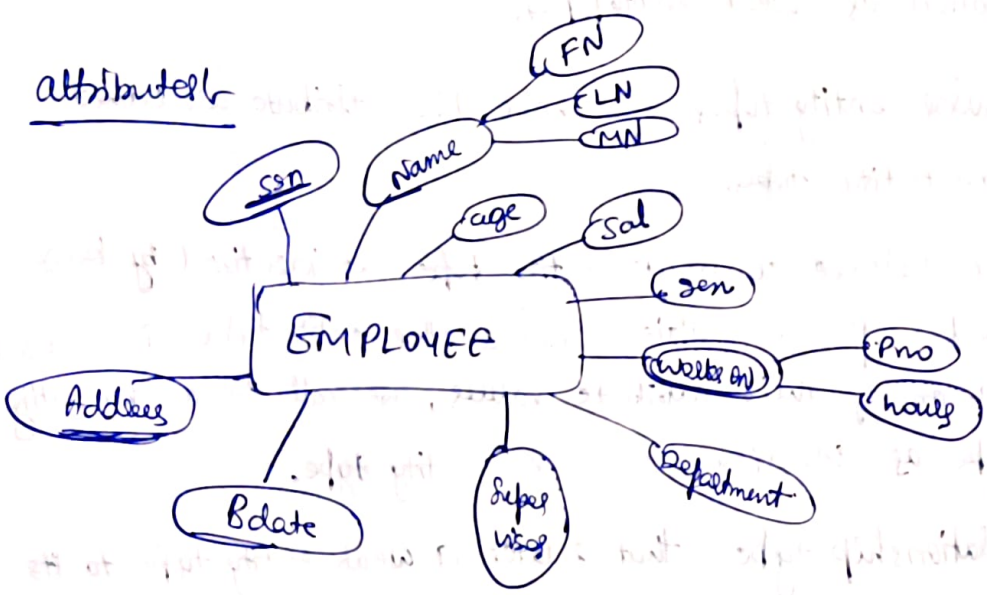
Steps In ER design :-

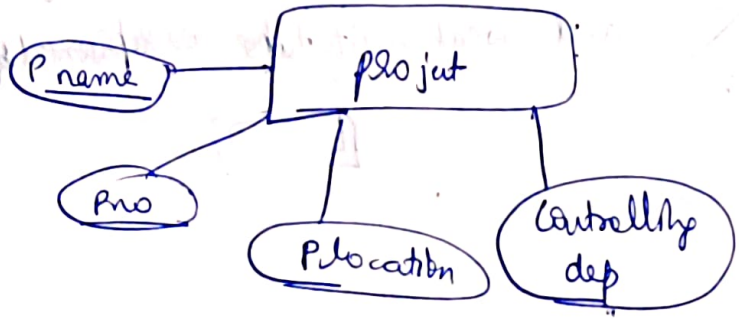
i] Identification of Entity types and attributes.

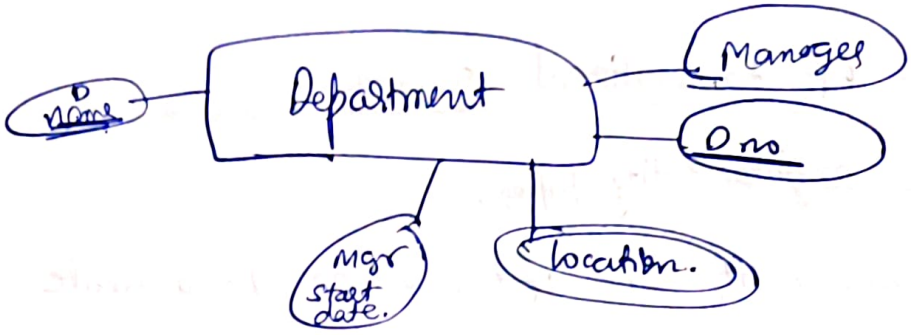
Entity types:

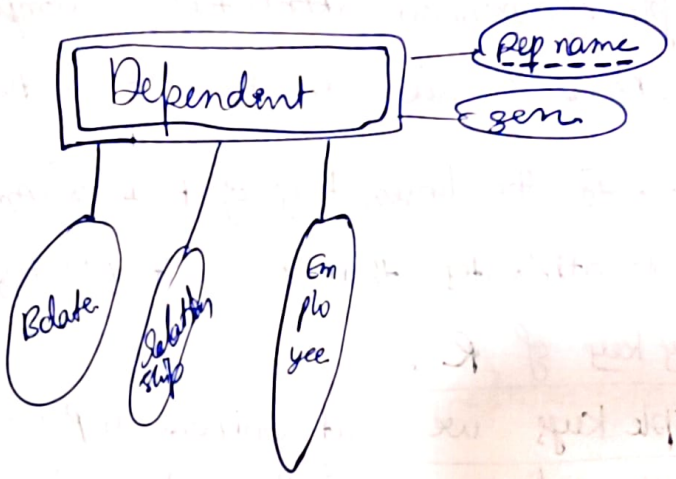


attributes:

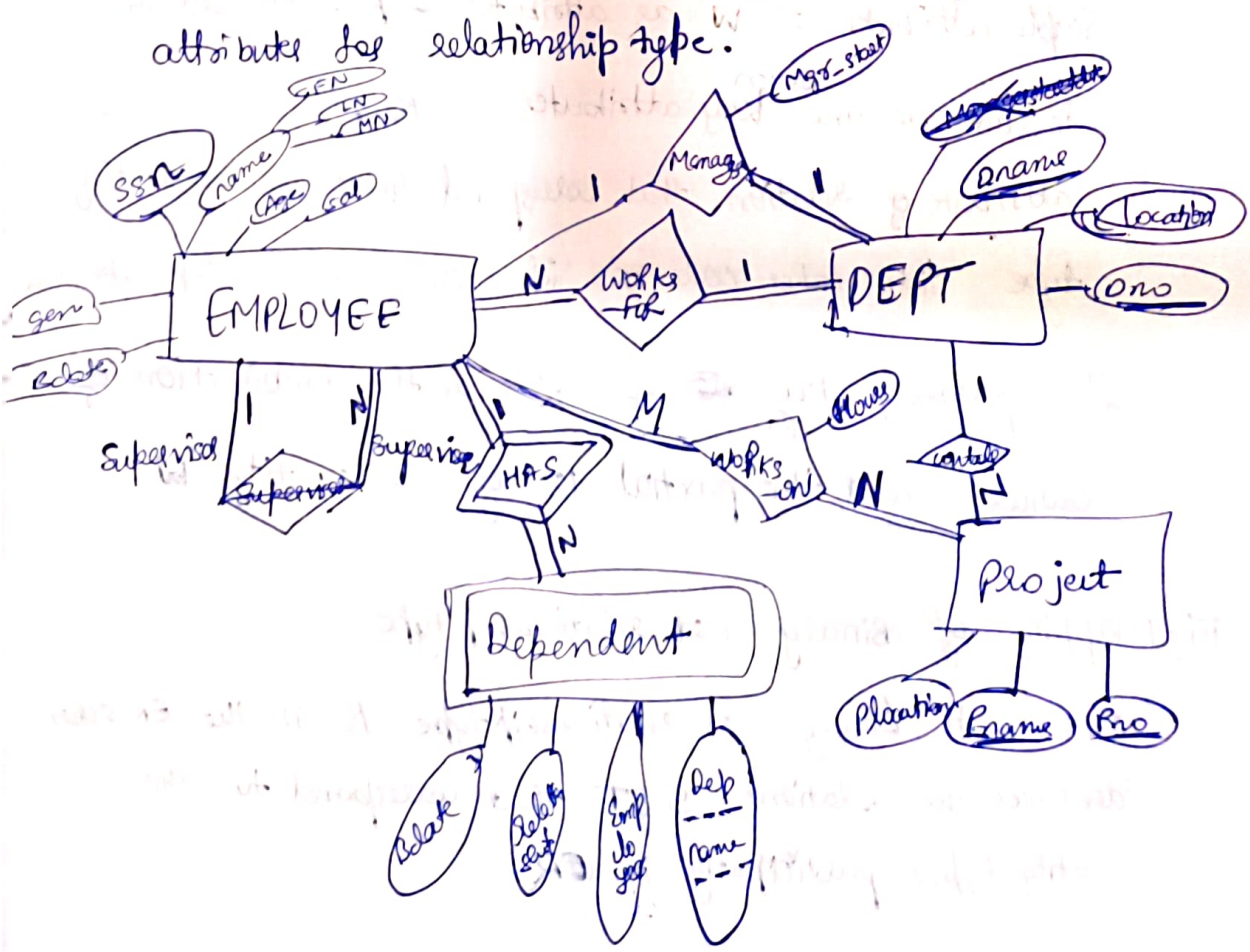








ii] Identification of relationship attributes and replacing relationship attributes with relationship types, Finally, Identify attributes for relationship type.



Mapping from ER to relational schema

Steps: i] Mapping of regular entity types:

For each regular entity type 'E' in the ER schema create the relation R that includes all the simple attributes of E include only a simple component attributes of a composite attribute. choose one of the key attributes of E as the primary key of R. If the chosen key of E is a composite then set of simple attributes that follow it will together form the primary key of R.

Note: When have multiple keys we select only one as primary key remains are candidate keys

ii) Mapping of weak entity types:

For each weak entity type 'W' in the ER schema with owner entity type E create a relation R and include all simple attributes of W as attributes of R. In addition include as foreign key attributes of R. The primary key attribute of relation that correspond to the owner entity type this takes care of identifying relationship type also. The primary key of R, is the combination of owner and the partial key of weak entity W

iii] Mapping of Binary 1:1 relationship type

For each binary 1:1 relationship type R in the ER schema identifies the relations S & T that correspond to the entity types participating in ER

choose one of the relations say S that is participating totally and include as foreign key in S, the primary key of T.

iv] Mapping of Binary 1:N relationship type

For each regular binary 1:N relationship type R identify the relation S that represents the participating entity type at the N^th side of relationship type, include as foreign key in S. The primary key of relation T that represents other entity type participating in R

v] Mapping of Binary M:N relation type

For each binary M:N relation type R, create a new relation S to represent R., include as foreign key attribute in S, the primary key's of the relation that represent participating entity types. Their combinations will form the primary key of S

vi] Mapping of multivalued attributes

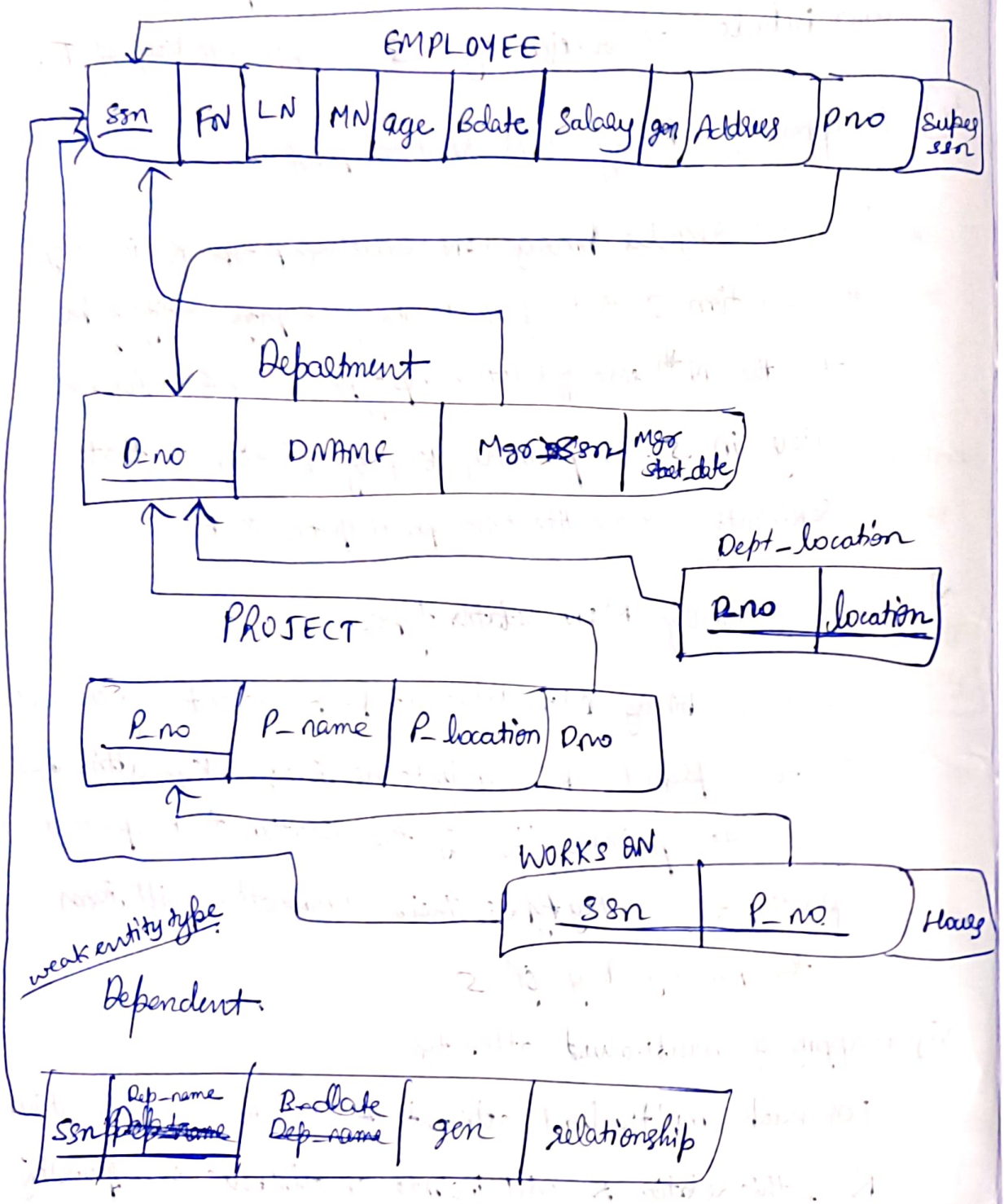
For each multivalued attribute A create a new relation R, this relation R will include an attribute corresponding A + the primary key attribute K as foreign key in R

Where K is the primary key of the relation that represents the entity type or relationship type that has A has an attribute.

Note:-

→ 1:N, 1:1 } No need of create relation

→ strong, weak, multivalued } need to create a relation


EMPLOYEE
Ssn
FN
LN
MN
age
Bdate
Salary
gen
Address
Dno
Super ssn
Department
D_no
DNAME
Mgr ssn
Mgr start date
Dept_location
Dno
location
PROJECT
P_no
P_name
P_location
Dno
WORKS ON
SSn
P_no
Hours
weak entity type
Dependent
Ssn
Dep_name
B_date
gen
relationship

Database Users :-

There are two different types of users

→ Actors on the scene
→ Workers behind the scene

i] Actors on the scene :

This include people who use the database in their day to day life

a] DBA [Database administrators]:

This involves people who are responsible for coordinating and monitoring various resources. DBA is also responsible for problems such as reach of security and poor system response time

b] DBD [Database designers]:

These people are responsible for identifying the data to be stored in the database and for choosing appropriate structure to represent and store the data, this is done before the implementation. These people also develop different views of the database. The final database design provided by designers must be capable of supporting the requirement of all user views.

Imp
** c] End Users :

End users are the people whose job requires access to the database for querying, updating and generating reports.

Following are the catogories of end user :-

i] Casual End users :

- These user occationally access the database but they may need different information each time
- They use sofasticated database query language to specify their requests

- Example: middle or high level managers

ii) ~~Second~~ Naive or parametric end users:

- The main job of these users is to constantly querying and update the database.
- They use standard types of queries and updates called canned transaction that are programmed and tested
- Example: Bank tellers, Reservation clerks for airline, hotels...

iii) Sophisticated end users:

This includes engineers, scientists and others who are familiar with facilities of the dbms in order to implement their applications

iv) Stand alone users:

This includes people who maintain personal databases by using readymade program packages that provide GUI's.

d) System Analysts and Application programs:

- System analysts determine the requirements of end users specially naive or parametric end users and develop specification for canned transactions, that meet these requirements.
- Application programs implement these specifications as programs then they test, debug, document and maintain these canned transactions, such analysts and programmers are commonly referred to as Software developers or software engineers.

- These people should be familiar with full range of capabilities provided by dbms to accomplish these tasks

2] Workers behind the scene:

These people are typically not interested in database itself but are associated with the design, development and operation of the dbms's software and system environment

This includes the following catogories:

i] DBMS designers and implementers:

These people design and implement the DBMS modes and interfaces as software package, modules include complex components like implementing catalog, processing query language, processing the interface, accessing and buffering data, controlling concurrency and finally handelling data recovery and security.

ii] Tool Developers:

These people design and implement tools which are optional packages that are often purchase separately. The packages include

a] packages for database design
b] Packages performance monitoring
c] Packages for natural language & graphical interface
d] Packages for prototyping
e] Package for simulation
f] packages for test data generation

iii] Operators and maintaince people:

These are responsible for the actual running and maintanance of the hardware and software environment for the database system.

Advantages of using DBMS approach:

i] Controlling Redundancy:

There are 3 different problems of redundancy

a] duplication of efforts

b] storage space

c] Inconsistancy

all these problems are reduced in database approach by integrating the views of different users groups during database design and by storing each logical data item in only one place in the database

ii] Restricting unauthorized access:

For this DBMS provides a security and authorization subsystem in which DBA uses to create account and to specify account restrictions. For example only the DBA staff may be allowed to use certain privilage softwares such as the software for creating new accounts

iii] Providing persistent storage for program objects:

Object oriented database management system are compatible with programming languages such as C++ and Java and hence avoid impedence mismatch problem (The data structures provided by DBMS are incompatible with programming language data structures). ~~Hence~~

Hence complex objects in c++ can be stored permanently in object oriented DBMS such objects are called as persistent objects.

iv] providing storage structures for efficient querying process :

Database system provides capabilities for efficiently executing query and updates.

- query processing and optimization module of DBMS is responsible for choosing an efficient query execution plan for each query based on existing storage structures.

- Database system also provides auxiliary files called indexes inorder to speedup disc search for a given query.

- DBMS also provides a buffering module that maintains parts of the database in main memory buffers

v] providing Backup and recovery :

DBMS provides backup and recovery subsystem which is responsible for recovering from hardware and software failures

vi] providing multiple users interfaces :

As many types of users with varying levels of technical knowledge use a database. A DBMS provides various interfaces such as

i] Query languages for casual users

ii] programming language interfaces for application programs

iii] Forms and command codes for parametric users

iv] Menu driven interfaces and Natural language interfaces for stand alone users

vii] Representing complex relationships among data :

DBMS have the capability to represent a variety of complex relations among the data, to define new relations and to retrieve and update related data easily and efficiently using concepts like foreign key

viii] Enforcing Integrity constraints :

DBMS provides capabilities for defining and enforcing integrity constraints like entity, domain, notnull, unique and referential integrity constraint.

ix] Permitting Inferencing and actions using rules :

- Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts. such systems are called as deductive database systems
- In a traditional DBMS an explicit procedural program code have to be written to support such applications where as in todays relational database systems rules are enforced using triggers and stored procedures
- More powerful functionality is provided by active database systems which provide active rules that can automatically initiate actions when certain events and conditions occured

x] Additional implications of using database approach

Database approach provides additional benefits such as

i] reduced application development time.
ii] Flexibility (change the structure of database as requirement changes)
iii] Availability of uptodate information
iv] Economies of scale.

When not to use a DBMS ?

i] The overhead cost of using a dbms are due to the following

i] High intial investment in hardware, software and training

ii] The generality that DBMS provides for defining and processing the data

iii] The overhead for providing security, concurrency control recovery and integrity functions.

Hence it may be more desirable to use regular files under the following circumstances (Do not use a DBMS under the following circumstances):

i] Simple well defined database applications that are not expected to change

ii] Real time requirements for some programs that may not be met because of the DBMS overhead

iii] No multiple users access to data

Retrival query in SQL :

A retrival query in sql consists of upto 6 clauses as follows :

```
select  <attribute and function list>
From    <table list>
[Where  <condition>]
[GROUP BY  <grouping attributes(s)>]
[Having  <group condition>]
[order by  <attribute list>];
```

Nested queries: A query is inserted within the Where clause of another query called as nested query.

The query enclosed within where clause is called inner query.

Ex: select name
from movie
where name not in (select name from crew);

- In where clause we have two ~~value~~ clauses:

in clause : used to compare values with set of values.

- inner query runs first, a query executed only once.

- not-in clause : used to compare a value with set of values and selects negation.

- When you are apply set theoritic like union, intersection and difference, make sure that the relations on which set theoritic operations are applied must be union compatible.

union compatibility: Two relations R and S are said to be union compatible

i] The degree of R and S must be equal (degree of a relation is the number of attributes of that relation)

ii] Let R is $A_1, A_2 \ldots A_n$ and S is $B_1, B_2 \ldots B_n$ then domain(A_i) = domain(B_i) $\forall$ $1 \leq i \leq n$.

minus:

Example:

(select name from studio) minus (select name from crew);

Aggregate functions: count(), max(), min(), AVG().

Selection condition: The condition which select a particular tuple.

Where clause :- It applicable to all tuples

Having clause :- It applicable to group by clause and must have necessary condition

Correlated nested queries :- Where class of inner query refers to an attribute of outer query such queries are called correlated nested queries.

difference between nested & correlated nested query : for each
- tuple of outer query the inner query executes each times once in correlated nested queries.

Where in nested queries executes inner query only once irrespective of tuples.

- Ex :-

```
Select  O.Oname
From    owner   O
Where   not exists ( select * from dog d
                     where  O.ssn = d.ssn );
```

13/11/24

DBMS Interfaces

→ Menu Based interfaces web clients or Browsing:

These interfaces present the user with the list of options called menus which avoids the user to memorize a specific commands and syntax of a query language

pull down menus are very popular techniques in web based user interfaces

→ Form based Interfaces:

A form based interfaces display a form to each user, users can fill out all the form entries to insert the new data or they can fill out only certain entries in which case the DBMS will retrieve matching data for the remaining entries

For an example - SQL* forms is a form based language supported by oracle

→ Graphical User Interfaces:

A GUI typically display a schema to user in diagrammatic form a user then can specify a query by manipulating the diagram. GUI utilizes both menus and forms in conjuction with pointing devices such as mouth.

→ Natural language interfaces:

These interfaces accept request written in english or some other language and attempt to understand them.
It interprets the query in natural language and generates a high level query corresponding to natural language request and submit it the dbms for processing.

→ speech input and output:

Here the request query and answer to a request will be in the form of speech, few applications with limited vocabularies such as Enquiries for telephone directory, flight arrival or departure, bank account info and so on, makes use of speach input and output.

→ Interfaces for parametric users:

These interfaces provides a small set of abbreviated commands for query request with the goal of minimizing the number of key stroke required for each request.

For example, functional keys used by a bank teller can we programme to intiate various commands

→ Interfaces for the DBA

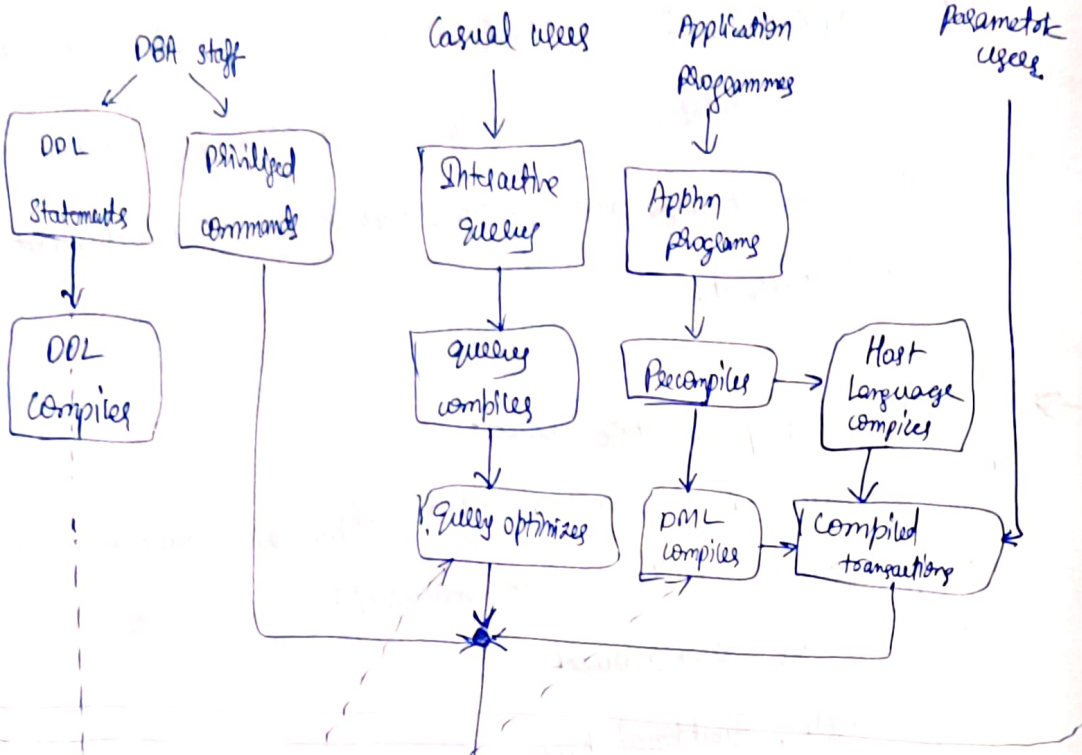
For Most database systems contain previlized commands that can be used only by DBA staffs. These include commands for creating accounts setting system parameters, granting account authorization, reorganizing the storage structure of database and so on.

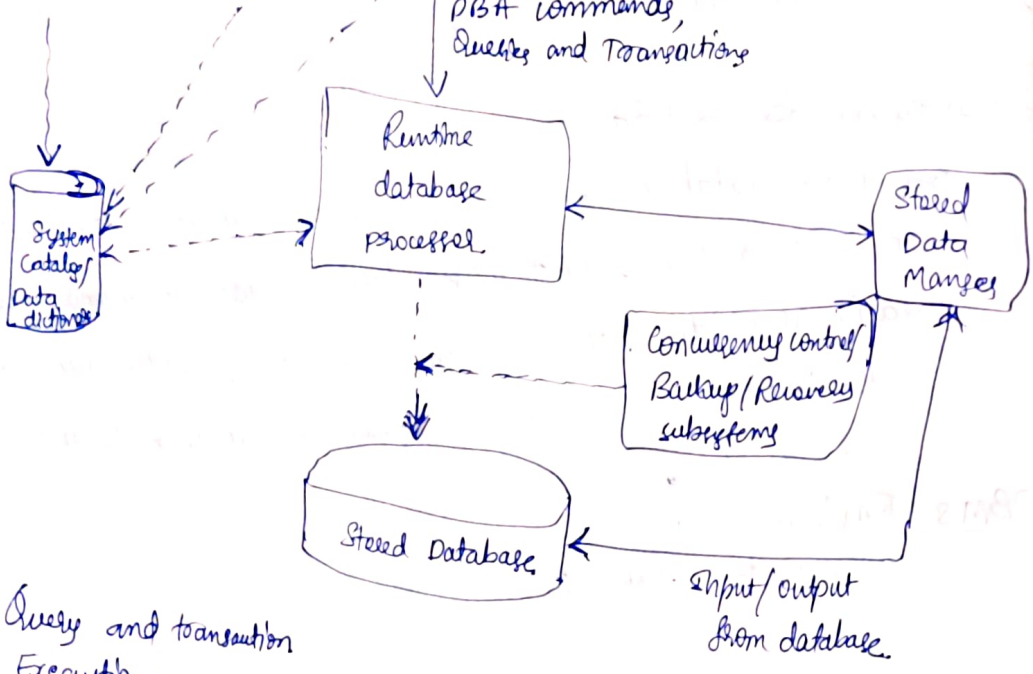
DBMS Environment:-

DBMS components Modules:

PTO

Users:





Query and transaction Execution:

→ Database system Utilities : [functions of database utilities]

These utilities help the DBA in managing the database system common utilities are the following types of functions :

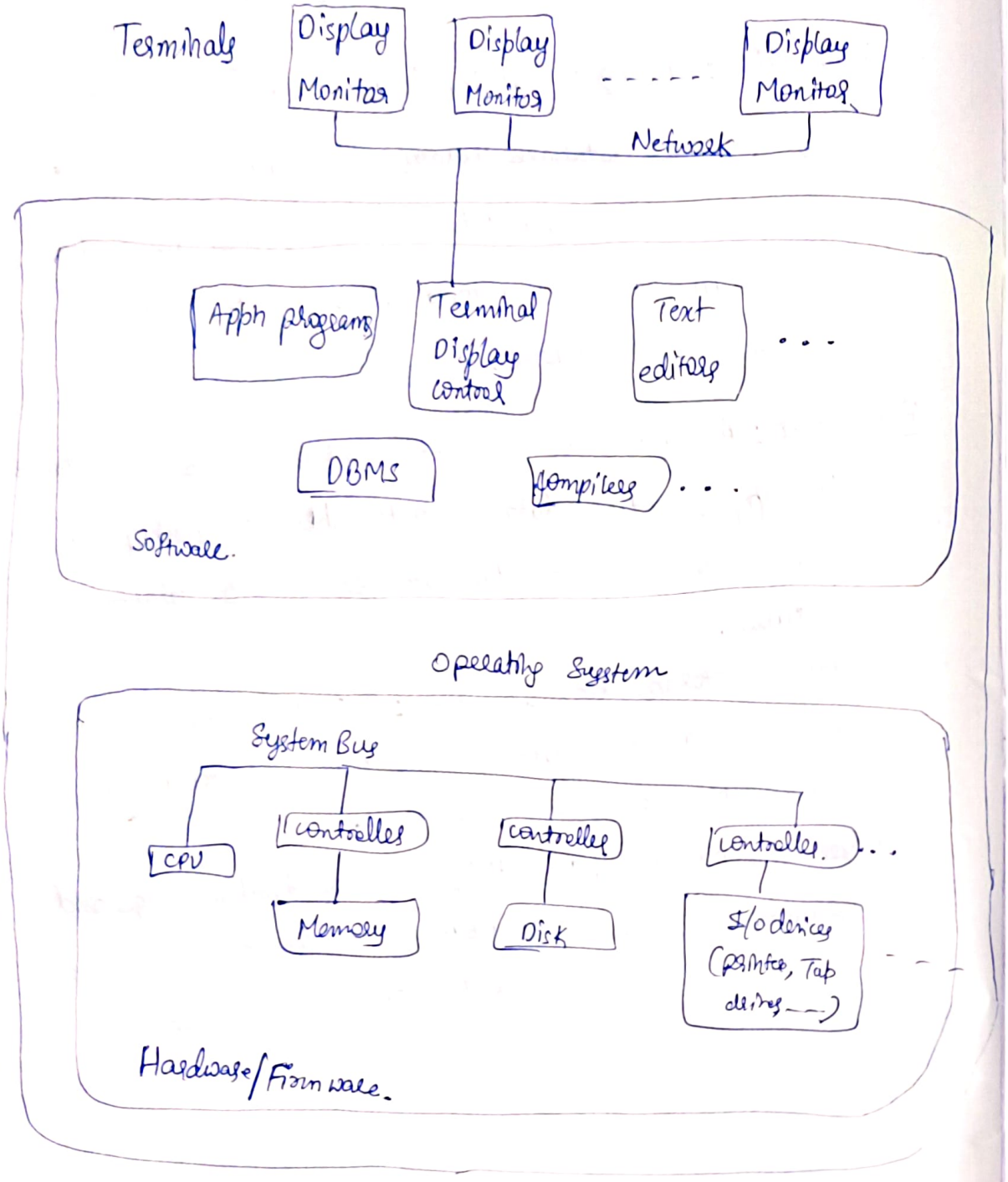
i] Loading : A loading utility is used to load existing data files into the database usually a sound source format of database and target database file structure specified to the utility, which automatically reformats the data and stores it in database.

ii] Backup : A backup utility creates backup copy of the database by dumping entire database onto tape. The backup copy can be used to restore the database incase of catastrophic failures.

iii] Database storage reorganization : This utility can be used to reorganize a set of database files into a different file organization to improve performance.

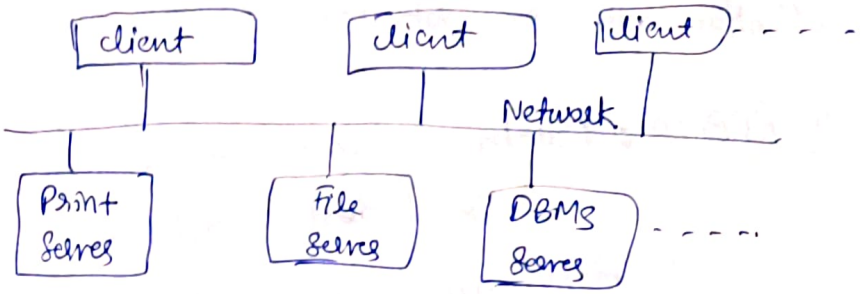
iv] Performance Monitoring : This utility monitors database usage and provides statistics to the DBA, which helps in making decisions such as whether or not to reorganize the files

Centralized DBMS architecture:



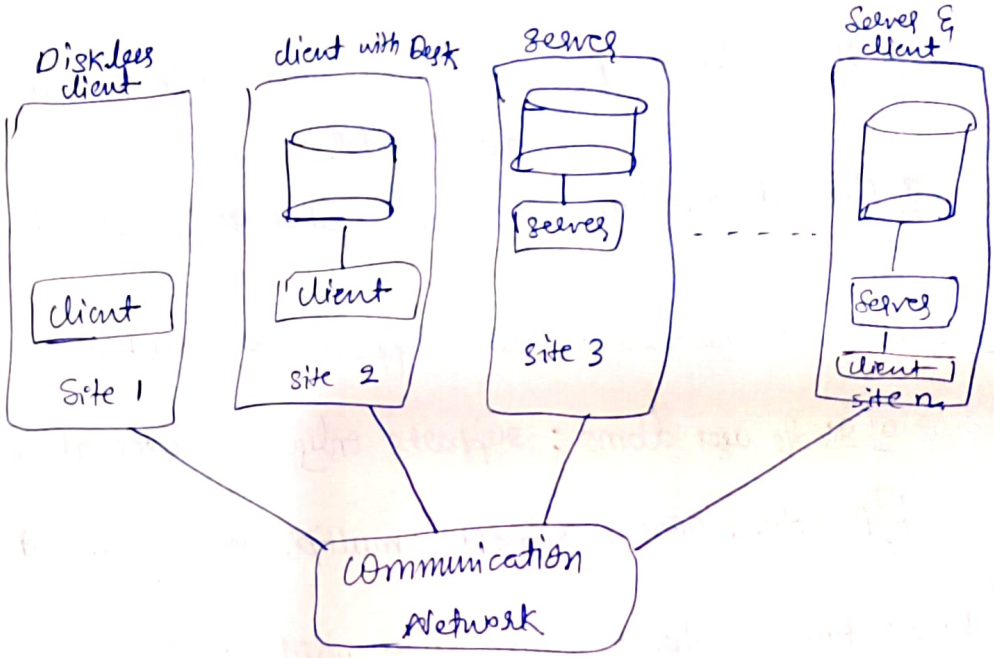
client/server architecture for DBMS:

→



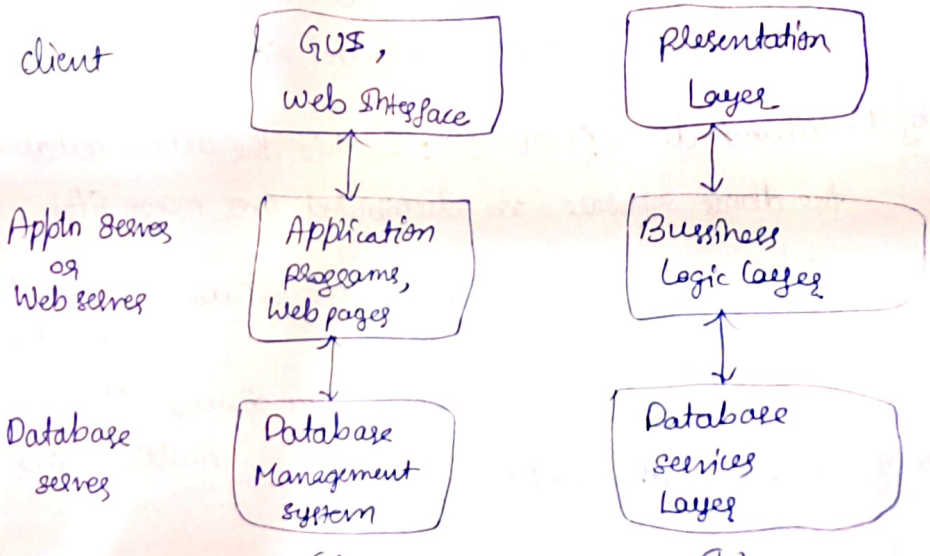
'Logical 2-tier client/server' architecture'

→



'physical two-tier client/server architecture'

→



'Logical 3-tier client/server architecture'

Classification of Database management system:

i] Classification based on data model

a] RDBMS : Database management systems that makes use relational data model (This data model used in many current, commercial database management systems)

b] OODBMS : DBMS's that makes use of object oriented data model

c] Heirachecal DBMS : which makes use of heirachecal data model

d] Network database MS : which makes use of network data model.

ii] Based on numbers of uses supposted by system :

a] Single uses dbms : supposts only one uses at a time

b] Multi uses DBMS : supposts multiple uses concurrently.

iii] Based on numbes of sites ovres which database is distributed :

a] Centralized dbms : In this the data is stored at a single computer a site, similarly dbms and database resides totally on a single computes site.

b] Distributed dbms (DDBMS) : In this the actual database and the dbms software are distributed ovres many sites.

types :- ↳ Homogenous DDBMS - It uses the same dbms software at multiple sites

↳ Hetegenous DDBMS - It uses different dbms software at multiple sites.

iv] Types of access path used :

v] Based on Usage

↳ general purpose : It can be used to develop any kind of applications

↳ Specific purpose : It can be used to develop specific kind of applns.

Relational data model and Relational data mode constraints :-

In the formal relational model terminology a row is called as tuple, a column header is called as attribute and table is called as relation. The data type describing the type of ~~columns~~ values that can appears in each column is represented by a domain of possible values. In general, a domain D is the set of atomic values each value in the domain is indivisable.

A relational schema R (relation intension) denoted by $R(A_1, A_2, ---A_n)$ is made up of a relation name R and list of attributes A_1 to A_n, each attribute A_i is the name of a role played by some domain D in the relational schema R. Domain of attribute A_i is denoted as dom(Ai). The degree of a relation is the number of attributes of the relation.

A relation state r of the relation schema R denoted as $r(R)$ (Relation extension) is a set of tuple $r = \{t_1, t_2, t_3 --- t_n\}$

where each tuple is an ordered list of n values $t = \{v_1, v_2, v_3 ---- v_n\}$ where each v_i and dom(A_i).

Mathematically a relation state can be written as

$$r(R) \subseteq \{ dom(A_1) \times dom(A_2) \times \cdots dom(A_n) \}$$

Characteristics of Relations:

1] Ordering of tuples in a relation: A relation is defined as a set of tuples.

Mathematically elements of a set have no order among them. hence tuples in a relation do not have any particular order. In other words a relation is not sensitive to the ~~ordinary~~ ordering of tuples.

Tuple ordering is not part of of o relation definition ∴ relation attempts to facts at a logical or abstract level, hence there is no preference for one logical ordering over another.

2] ordering of values within a tuple :- According to the definition of a relation, an n-tuple is the ordered list of n-values so that the ordering of values in tuple and hence of attribute in a relational schema is important. However at the logical level the order of attributes and their values is not that important as long as the correspondance between attributes and values maintained.

3] Nulls in tuples:- An attribute value can be represented using NULL and will be interpreted by the relational model in any of the following case:

a] The value of the attribute is unknown
b] The value of the attribute is not applicable
c] The value of the attribute is missing.

4] Interpretation of a relation: Some relation may represents facts about entities where as other relations may represent facts about relationships.

Relational Model Notations:

1] A relational schema R with degree n is denoted as $R(A_1, A_2, A_3 - - - A_n)$

2] Capital letters like Q, R, S denote relation names

3] Small letters like q, r, s denotes relation states

4] Small letters t, u, v denotes tuples.

5] In general the name of a relation schema such as STUDENT indicates relation ~~such as~~ state also, where as STUDENT(name, ssn, - - -) refers only to the relation schema.

6] An attribute A belongs to relation R is denoted by R.A

7] An n-tuple t in a relation r(R) is denoted by $t = <v_1 v_2 v_3 - - - v_n>$ where v_i and dom(A_i), then $t[A_i]$ or $t.A_i$ refers to the value v_i in t for attribute A_i

Relational model constraints

constraints on databases are generally divided into 3 main categories

i] Inherent model based or implicit constraints :
The constraints that are inherent in the data model

ii] Schema based or explicit constraints :
constraints that can be directly expressed in schemas of the data model using DDL

iii] Application based or semantic constraints / business rules :
constraints that cannot be directly expressed in the schema's of data model and hence must be expressed by the application programs.

→ Schema based constraints

a] Domain constraints : Domain constraints specify the within each tuple value of each attribute A must be an automic value from the domain of dom (A)

b] Key constraints : Usually some subsets of attributes of a relation schema R with the property that no two tuples in any relation state r(R) should have the same combination of values for these attributes.

— Suppose that one such subsets of attributes be. SK then any two distict tuples t_1 and t_2 in relation state r(R) we have the constraint that $t_1(SK) \neq t_2(SK)$.
Any such set of attributes SK is called a super key of relation schema R.

- A super key SK specifies the uniqness constraints that no two distict tuples in any relational state r(R)

can have same value as SK.

- Every relation has atleast one default superkey that is set of all its attributes

- A key K of relation schema R is a superkey of R with the ~~read~~ additional property that removing any attribute from K, leaves set of attributes K' that is not a superkey of R anymore, hence a key satisfies two constraints

  → two distict tuples in any state of the relation cannot have identical values for the attributes in key,

  → It is a minimal superkey that is a superkey from which we cannot remove any attributes and still have uniqueness constraint in condition ~~one~~ (i) hold.

In general the relational schema may have more than one key in this case each of the key is called as candidate key, It is common to designate of one of the candidate key as primary keys of the relation.

c] Not null constraint

This constraint restricts null values for the attributes

d] Integrity constraints

These constraint are enforced on database schema to maintain integrity of database.

There are 2 types of integrity constraints:

→ Entity integrity constraint: It states that no primary key value can be null

→ Referential Integrity constraint: It is specified between two relation and is used to maintain the consistency among tuples in the two relations.

- Informally it states that a tuple in one relation refers to another relation must refers to an existing tuple in that relation
- To define this constraint formally
- This constraint is enforced using foreign key which specifies referential integrity constraint between two relational schemas R_1 and R_2.
- A set of attributes FK in relation schema R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following rule.
  - → The attributes in FK have the same domain as the primary key attributes PK of R_2 then the attributes FK are said to reference (or) refers to the relation R_2.
  - → A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuples t_2 in the current state $r_2(R_2)$ or is null

    if $t_1[FK] = t_2[PK]$ we say that tuple t_1 references are refers to the tuple t_2
  - → In this defination R_1 is called referencing relation R_2 is called referenced relation. If these 2 conditions hold a referential integrity constraint from R_1 to R_2 is said to hold

update operations and dealing with constraint violations :

i) ~~Insert~~ :- There are 3 different types of update operations :

i) Insert
ii) Delete
iii) update

constraints violated by Insert operation

Domain constraints : if ssn is int then if we enter characters then again violation of domain constraints.

NOT NULL :- The ssn is primary key it can be null hence not null constraint violated.

UNIQUE : all values are unique, ssn is unique we cannot enter same value hence unique also violated by insert.

Key constraints : If unique constraint violated then violation of key constraints.

Entity integrity constraints :

Referential integrity constraints :

Emp ssn	DEP Mgr ssn
1	
2	
3	4

violating the referential integrity constraints.

The above constraints also violated by update operations.

- constraints violated by delete operation :-
  - Domain constraints : It is not violated
  - Not NULL : No violation
  - Unique : No violation
  - Key constraints : No violation
  - ESC : No violation
  - RIC : It is violated

→ Select the ssn of all employees who work the same (project, hours) combination on some project that employee that employee whose ssn=1234 works on.

```
Select ssn from works_on
  where (Pno, hours) IN (select Pno, hours
                         from works_on
                         where ssn='1234');
```

→ Retrieve name of employees whose salary is greater than salary of all the employees in department 5

```
Select name from employee
  where ~~dno=5 and~~ salary >= (select max(salary) from
                                 employee where
                                 d_no=5)
```

(OR)

```
Select name from employee
  where salary >all( select salary from employee
                     where dept=5);
```

→ Retrieve name of each employee who has a dependent with the same firstname, and same genders as the employee

```
Select e.FN, e.LN, e.Mn
       ~~e.name~~ from employee e, dependent d
  where (~~d.ssn=e.ssn~~ e.ssn = d.ssn) and e.first_name = d.~~first~~name
        and e.gender = d.gender
```

```
Select FN, LN, MN from employee e
where e.ssn in ( select d.essn from dependent d
                 where e.firstname = d.name
                 and e.gender = d.gender);
```

```
select FN, ML, LN from employee e
    where (FN, gender) in ( select name, gender
    from dependent d where e.ssn = d.essn);
```

Note: The exists function in SQL is used to check whether the result of a correlated nested query is empty or not.

The result of exists is a boolean value true / false

exists(Q) returns true if there is atleast one tuple in the result of nested query Q and return false otherwise.

In contrast not exists(Q) returns true if there are no tuples in the result of nested query Q and returns false otherwise.

```
Select FN, MN, LN from employee e
    where exists ( select essn from dependent d
                   where e.ssn = d.ssn
                   and e.fn = d.name
                   and e.gen = d.gen );
```

→ List the names of managers who have atleast 1 dependent.

```
select FN, LN, MN from employee e, department d
    where e.ssn = d.mgr_ssn
    and exists ( select ssn from dependent dl
                 where e.ssn = dl.ssn );
```

(OR)

```
select FN, LN, MN from employee e
    where exists ( select * from dep d
                   where e.ssn = d.mgr_ssn)
    and exists ( select ~~name~~ from dependent D
                 where e.ssn = D.ssn );
```

→ Retrieve names of employees who have no dependents

```
select FN, LN, MN from employee e
    where not exists ( select * from dependent d
                       where e.ssn = d.ssn);
```

→ Retrieve name of employee who works on all project controlled by dept number 5

```
select FN, LN, MN from employee e, ~~department D~~
X
    (select p_no from project
     where dno = 5);
```

```
Select name from Emp
where ( select Pno
        from works_on
        where ssn = Essn)
      CONTAINS
      ( select pno
        from proj
        where dno = 5);
```

⇒ contains clause. R contains S or R ÷ S (Division)

- Whenever in question if there will be all clause normally query will be solved using contains.

- If contains not supported

$$R \div S \iff (S - R) \text{ is empty}$$

S − R is empty ⟹

```
select name from Emp
where not exists ( (select P_no
                    from Proj
                    where dno = 5)
                   EXCEPT
                   ( select P_no
                     from works_on
                     where ssn = essn)
                 );
```

→ For each department retrieve the department number, the number of employee in the department and their average salary.

```
Select d_no, count(*), AVG(salary), from employee e,
department d where e.dno=d.dno;
```

(or)

```
select d_no, count(*), AVG(salary) from
emp group by dno;
```

→ For each project retrive project number, project name number of employees who work on that project.

```
select p_no, P_name, count(*) from project P,
works_on w where p.pno = w.pno,
group by Pno, Pname;
```

→ For each project on which more than 2 employees work, Retrieve project number, project name and the number of employees who work on that project.

```
select P_no, P_name, count(*)
from Proj P, works_on W
where P.Pno = w.Pno group by Pno, Pname
Having count(*) > 2,
```

→ For each project Retrieve project number, project name and number of employees from department 5 who work on the project.

```
select  P_no, P_name, count(*)
from proj P, works_on w, department d
where  P.Pno = w.Pno and
       dno = 5 ;
```

```
select  Pno, Pname, count(*)
from proj P, works_on w, employee E
where P.Pno = w.Pno  and E.ssn = w.ssn  and
      E.dno = 5
group by Pno, Pname ;
```

→ List the name of employees who have two or more dependents

```
select  name from employee e, dependent d
where.  e.ssn = d.essn and.
group by d.essn
having  count(*) >= 2 ;
```

(or)

```
select nam from employee
where ( select count(*) from dependent
        where  ssn = essn ) >= 2 ;
```

→ Find the sum of salaries of all employees maximum salary, minimum salary, average salary.

```
select sum(salary) as salary_sum,
       MAX(salary) as salary_max,
       MIN(salary) as salary_min,
       AVG(salary) as salary_avg.
from employee.;
```

Note: Aggregate functions:

These aggregate functions are used in select clause and having clause.

- COUNT – count the number of tuples.
- SUM
- MAX
- MIN
- AVG

→ Find the sum of salaries of employees of the research department as well as the max salary, min salary, avg salary.

```
select sum(salary), max(salary), min(salary),
       Avg(salary) from employee e, department d
       where e.dno = d.dno and
             dname = 'Research';
```

→ Retrieve total number of employee in the company

→ Number of employees in research department.

→ Select count(*) from employee ;

Select count(*) from employee e, department d
Where e.dno = d.dno
and dname = 'Research' ;

→ Count the number of distinct salary values in the database.

Select ~~distinct count(salary)~~ count(distinct salary) from employee ;

Join operations :-

Two types of join operations.

- Inner
  - θ join
  - Equi join
  - Natural
- Outer
  - left
  - Right
  - full

Inner join operations list out only the matched tuples after applying relations.

left :- match in tuples of left and unmatched tuples of right relations

right :- match in tuples of right and unmatched tuples of left relations.

θ join : {>, ≥, <, ≤, =, !=} set of comparision operators If join condition contains these operators we called as θ join.

Equijoin : Particulars the operators used is = then we called as Equi join

Natural join : When two attributes have same name in both tables.

List the name and address of employees whose belongs research dept.

→ ~~List~~ select ~~the~~ name, address from employee e, ~~Join~~ dept D

where e.dno = D.dno and dname = 'Res';

(or)

by using join

select name, address from employee join dept d on e.dno = d.dno where dname = 'res';

(or)

when they have same name

select name, address from Emp Natural join dept where dname = 'res';

(or)

When having different name:

```
Select name, address from employee
    Natural join (dept as de (dno, dname, mgr, st))
    where dname = 'research';
```

→ List name of employee and its supervisor:

```
Select e.name, s.name from employee e, employee s
    where e.super_ssn = s.ssn ;
```

+ = ⟶ left outer join

= + ⟶ right outer join

+ = + ⟶ full outer join

```
Select e.name, s.name from employee e, employee s
    where e.super_ssn += s.ssn ;
```
(left)

(or)

```
Select e.name, s.name employee e, ~~employee s~~
    left outer join employee s on e.super_ssn = s.ssn;
```

left join (in some versions)

→ For every project located in stanford list the pno, controlling dno and department manager name and address.

select pno, dno, name, address

from (project join department on P.dnum = d.dno)

join emp on d.mgr_ssn = E.ssn

where plocation = 'stanford';

→ ~~For each department retrieve department num~~

Retrieves ssn's of all employees who work on project numbers 1, 2 or 3.

select ssn from ~~employee e, project~~ Works_on

where P_no = 1 or Pno = 2 or Pno = 3;

(or)

select ssn from works_on

where P_no in (1, 2, 3);

Datatypes | Domains SQL :-

i] Numeric :-

INT (or) INTEGER

SMALLINT

FLOAT, REAL

DOUBLE PRECISION

Decimal (i, j)

Dec (i, j)

Numeric (i, j)

} i is the precision that is the total numbers of decimal digits,
j is the scale that is the numbers of digits after decimal points

ii] Character string :-

char (n)

varchar (n)

One more datatype called BLOB [Binary large objects]

iii] Bit string

BIT (n)

Whenever the bit string value preceded with B'1010'

iv] Boolean

true / false / unknown it will store

v] Date - 'YYYY-MM-DD'

vi] time - 'HH:MM:SS'

vii] timestamp : This includes the date and time fields + a minimum of 6 positions for decimal fraction of seconds

These timestamp data preceded with TIMESTAMP

PTO

These timestamp data preceded with

TIMESTAMP 'YYYY-MM-DD HH:MM:SS-----'

viii) Interval

This specifies a relative value that can be used to increment/decrement an absolute value of a date time/timestamp.

check constraint can used in DDL statements

Ex: Dnum int check (Dnum >=1 and Dnum <=10);

User defined datatype

Create Domain

DNUM int check (DNUM >=1 AND DNUM <=10);

Dno DNUM ;

Referential triggered actions :-

The default action that sql takes for an integrity violation is to reject the update operation that will cause a violation. However the schema designer can specify an alternative action to be taken if a referential integrity constraint is violated by attaching a referential triggered action class to any foreign key constraint.

→ This options include set null, [illegible], [illegible]

these options must be qualified with either on delete or on update .

Ex:

(Emp) → dno [In employee table]

foreign key (Dno) references Department (Dno) on update cascade;

foreign key (Dno) references Department (Dno) on delete set null;

(or)

set default;

Ex: to remove default constraints using alter statement

Alter table tn alter column cn drop default;

(or)

age int constraint Dcn Default 20;

Alter table tn drop constraint Dcn; (by specify constraint name)

clauses in SQL :-

select
from
where
group by
Having
order by
limit

13/12/24

Procedures and triggers :-

procedure : These are the functions, stored procedures in sql

→ create procedure display_view (parameters)

parameters

→ IN parameters - passing parameters to function
→ OUT parameters - returning to the calling function

→ To calling the procedure

call display_view ('parameters');

- default delimeter in sql is ; semicolon.

- In procedures .. we need to change the default delimeters to other delimeters like //

delimeter //

procedure

//

Cursors :- A pointer which is pointing to table, as when each row retrieve automatically it moves next tuples untill last row.

Triggers :- It is also a stored procedure.

need to go for triggers :- In procedures the whenever constraints get voilated it will not take any action.

- need not to ~~call~~ explicitly ~~the~~ call triggers
- triggers only we can call for DML's

two types :-

→ trigger execution before DML. → before – Insert, update, delete

→ trigger execution after DML → after – Insert, update, delete.

NEW - new specifies
OLD - old specifies

DML commands :-

Insert values from existing table

→ Insert into works_on_info (Emp_name, proj_name, Hours_per_week)

```
select E.Lname, P.Pname, W.Hours
from Project P, works_on W, Employee e
where P.pnumber = W.Pno and W.Essn = E.ssn;
```

→ create table works_on_info

```
( Emp_name varchar(15),
  proj_name varchar(15),
  Hours_per_week decimal(3,1));
```

→ Delete from tablename;

→ update employee
set salary = salary * 1.1
where Dno=5;

→ update project
set plocat = 'Bellaire', Dnum=5
where Pnumber=10;

Assertions:

A constraint that cannot be created using DDL statements (that is constraints which are not schema level constraints) can be created using assertions.

For example to specify the constraint that the salary of an employee must not be greater than the salary of managers of the department that the employee works for we can create an assertion for this as follows.

→ create assertion salary_constraint
check (NOT EXISTS (select * from Employee e, employee m, department d
where E.salary > M.salary and
E.Dno = D.Dnumber and
D.Mgr_ssn = M.ssn));

- Assertions also a stored procedure and they are default action will be rejection.

Views:-

A view in sql is a single table that is derived from other tables these other tables can be base tables or previously defined views. A view does not necessarily exists in physical form and hence it is considered as a virtual table in contrast to base tables whose tuples are actually stored in the database.

V1 :

```
create view  works_on1
as select  Fname, Lname, Pname, Hours
from  employee, Project, works_on
where ssn=Essn and Pno=Pnumber;
```

V2 :

```
create view  dept_info (Dept_name, No_of_emps, Total_sal)
as select  Dname, count(*), sum(salary)
from  department, employee
where  Dnumber=Dno
group by Dname;
```

Schema of view

works_on1

Fname	Lname	Pname	Hours

Dept_info

Dept_name	No_of_emps	Total_sal

→ create view viewname;

→ drop view viewname;

Qv1 : Select Fname, Lname
From works_on1
Where Pname = 'ProductX';

Note : Whenever the view contains aggregate functions the update is meaningless.

View Implementation :

Two main approaches for implementing view

i] Query modification

ii] View materialization

i] Query modification :-

It involves modifying view query into a query on the underlying base tables

ii] View materialization :-

It involves physically creating a temporary view table when the view is first query and keeping that table on the assumption that other queries on the view will follow, in this case an efficient strategy for automatically updating the view table when the base tables are updated must be developed in order to keep view up to date, this is done using incremental update that is the view is generally kept as long as it is being queried. If the view is not queried for a certain period of time the system automatically removes the physical view table and recompute it

from scratch when future queries reference the view.

View updation:-

Updating of views is complicated and also ambigous. in general an update on a view defined on a single table without any aggregate functions can be mapped to an update on the underlying base table under certain conditions. In other situations where a view involving joins leads to ambiguity.

updating views

UV1 : update works_on

```
set    Pname = 'Product Y'
where  Lname = 'smith' and Fname = 'John'
       and Pname = 'ProductX';
```

Updating the base table :

✓ correct updation

```
update works_on
set    Pno = (select Pnumber from project where
              Pname = 'ProductY')
where  Essn IN (select ssn from employee
                where Lname = 'smith' and Fname = 'John')
       and
       Pno = (select Pnumber from project
              where Pname = 'ProductX');
```

✗ wrong updation

```
update project set pname = 'ProductY'.
where  Pname = 'ProductX';
```

Consider update operation as follows:

```
update  Dept_Info
    set  Total_sal = 100000
    where  Dname = 'Research';
```

This update operation does not make sense because total salary defined to be the sum of individual employee salaries.

In summary we can make the following observations with respect to views.

i) A view with a single defining table is updatable

ii) If the view attributes contain the primary key of base relation as well as all attributes with the not null constraint that do not have default values specified

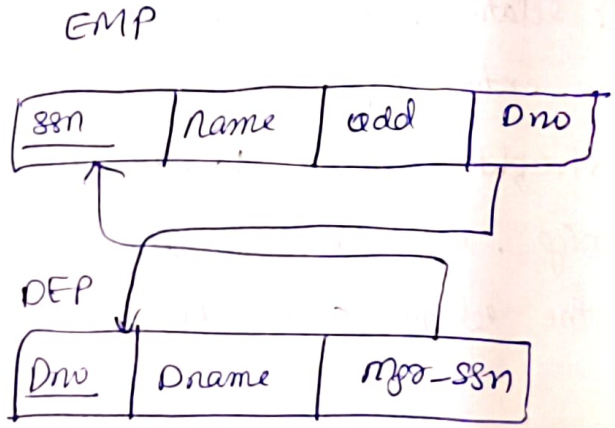
ii) Views defined on multiple table using joins are generally not updatable.

iii) Views defined using grouping and aggregate functions are not updatable

Informal design guidelines for relation schemas:-

1] Semantics of attributes: Whenever we group attribute to form a relation schema we assume that attribute belonging to one relation have certain real world meaning and proper interpretation associated with them. The semantics of relation refers to the interpretation of attribute values in a tuple. In general the easier it is to explain semantics of the relation the better the relation schema design will be,

For example Let us consider the following two relations



Here each tuples of employee relation represents an employee with the values for name, ssn, address and the number of the department that the employee works for.

The department number attribute is a foreign key that represents an implicit relationship between employee and the department.

Similarly the semantics of the department relation is clear.

Let us consides the following relation schema that combines the attributes of employee and department relation

EMP_DEP

Ssn	name	add	Dno	Dname	mgr-ssn

Even though the above relation have the clear semantics, it is the result of bad design as it combines the attributes from different relations.

→ Guiddine 1 :- Design a relation schema so that it is easy to explain its meaning, do not combine attributes from multiple entity types and relationship type into a single relation. If a relation schema corresponds to one entity type or one relationship type, it is straight forward to interpret and explain its meaning. Otherwise semantic ambuigities will result and the relation cannot be easily explained.

2 Redundant information in tuples and update anamolies

One goal of schema design is to minimize the storage space used by the ~~various~~ base relation. Grouping attributes into relation schema's has a significant effect on storage space.

For example, in EMP_DEP the attribute values pertaining to a particular department are repeated for every employee who works for that department.

Another problem with using the relations like employee_dept is update anamolies. These are classified into insertion anamolies, deletion anamolies, modification anamolies.

i] Insertion anamolies:-

Insertion anamolies can be differentiated into two types, and can be illustrated using emp_dept relation as follows:

a] To insert a new employee tuple emp_dept we must include either the attribute values for the department that the employee works for or null if the employee does not work for a department.

Similarly to insert a new tuple for an employee who works for dno 5 we must enter the attribute values of department 5 correctly so that they are consistent with values for department 5, in other tuples emp_dep.

b] It is difficult to insert a new dept that has no employees as yet in emp_dept relation.

ii] deletion anamolies:-

If we delete from emp_dept, an employee tuple that happens to represent last employee working for a particular department, the information concerning that department is also lost from the database.

iii] Modification anamolies:-

In emp_dept if we change the value of one of the attributes of a particular department we must update the tuples of all employee who work in that department otherwise database will become inconsistent.

→ Guideline 2:

Design the base relational schema's so that no insertion, deletions are modification anamolies are present in the relation. If any anomolies are present note them clearly and make sure that the programs that update the database will operate correctly.

~~Guideline~~ 3: Null values in tuples:

* In some schema designs many of the attributes will end up with null values for all the tuples, this can waste space at the storage level and also lead to problems with understanding the meanings of attributes and with specifying join operations at the logical level.
* Another problem with null is how to account for them, when aggregate operation such as count or sum are applied.
* ~~And also~~ Similarly select or join operations involve comparisons, if null values are present the results become unpredictable as nulls can be interpreted in many ways such as

i] The attribute does not apply to this tuple
ii] The attribute value for this tuple is unknown
iii] The value is known but absent that is it has not be recorded yet.

→ Guideline 3: As far as possible avoid placing attributes in a base relation whose values may frequently be null. If nulls are unavoidable make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

~~Guidelines~~ Generation of spurious tuples

* Spurious tuples are those rows in a table which occur as a result of joining two tables inthe wrong manner. They are extra tuples that might not be required.
* If a relation is denoted by R and its decomposed relation are denoted by $R_1, R_2 - - - - - R_n$, then the condition for not getting any spurious tuples is denoted by

$$R_1 \bowtie R_2 \bowtie R_3 \bowtie - - - - \bowtie R_n = R$$

(join)

where as the condition for getting spurious tuples is denoted by $R \subseteq R_1 \bowtie R_2 \bowtie R_3 \bowtie - - - - \bowtie R_n$.

Let us consides the following example:

R

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

R_1

A	B
a_1	b_1
a_2	b_1

R_2

A	C
a_1	c_1
a_2	c_2

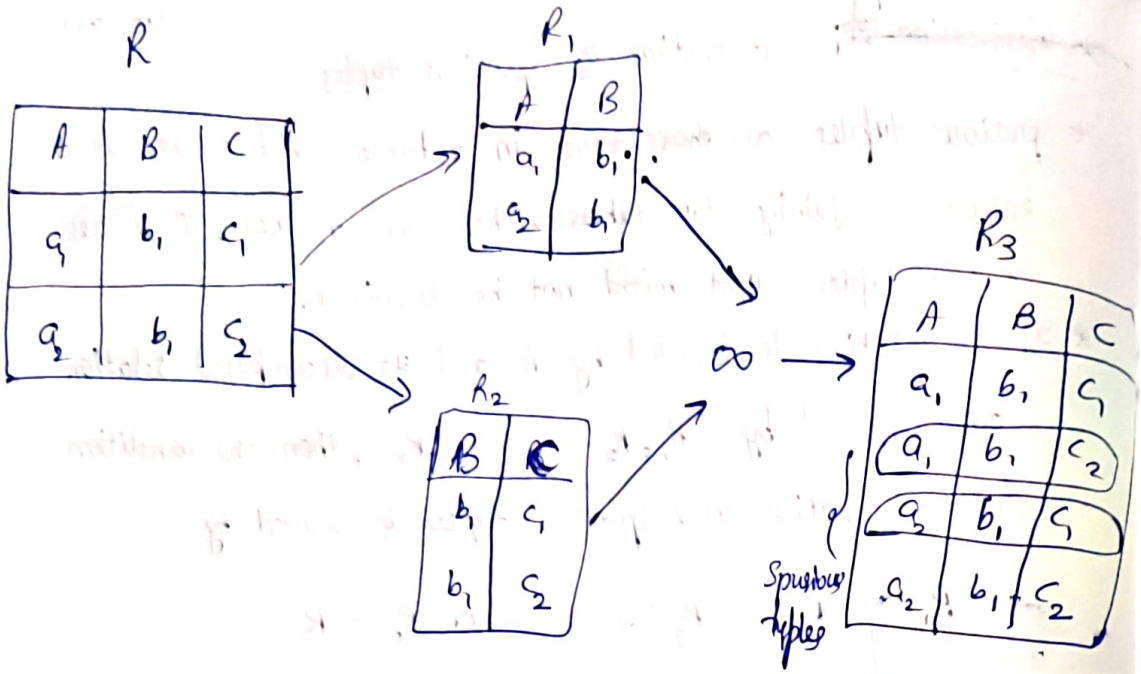
$\bowtie$

R_3

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

Note: The joined relation R_3 is equivalent to the original relation R and hence no spurious tuples are generated.

This kind of join is also called as loss-less join or non-additive



Note:-

Spurious tuples are tuples where those are not present in original relation R but they generated after join

These join is called as lossy join or additive

Now the resultant relation has two additional tuples which are not in the original relation and are called as spurious tuples.

→ Guideline 4:-

Design relational schemas so that they can be joined with equality conditions on attributes that are primary key foreign key in a key guarantees that no spurious tuples are generated. Avoid relations that contains matching attributes that are not primary key, foreign key combinations because joining on such attributes may produce spurious tuples.

NoSQL Database:-

→ No sql stands for Non SQL or Not only SQL or Non relational SQL databases

→ NoSQL is a type of DBMS that is designed to handle and store large volumes of unstructured and semistructured the data.

→ NoSQL databases are generally classified into four main categories

i] Document Databases: These databases store data as semistructured documents such as json or xml.

ii] Key value stores: These databases store data as key-value pairs

iii] Column family stores: These databases store data as column families which are sets of columns that are treated as single entity.

iv] Graph Databases: These databases store data as nodes and edges.

MongoDB

→ It is an open source cross platform document oriented NoSQL database.

→ It provides a language called MQL stands for Mongo DB query language.

Functional Dependencies :-

→ A functional dependency is a constraint between two sets of attributes from the database.

Let us consider an universal relational schema $R = (A_1, A_2 \text{---} A_n)$

→ A functional dependency denoted by $X \rightarrow Y$ (X determines Y) between two sets of attributes X and Y that are subsets of R is a constraint on the possible tuples that can form a relational state $r(R)$ that is for any two tuples t_1, t_2 belongs to $r(R)$ if $t_1[X] = t_2[X]$ then they must also have $t_1[Y] = t_2[Y]$

→ This means that the values of Y component of a tuple in R depend on or determined by the value of X component.

→ Alternatively values of X component of a tuple uniquely or functionally determine the value of Y component.

→ We also say that there is a functional dependency from X to Y or Y is functionally dependent on X.

→ FD or fd is the abbreviation of functional dependency

→ X is left side FD, Y is rightside of FD

The set of all dependencies that include F (given a set of functional dependencies) as well as all dependencies that can be inferred from F is called closure of F and is denoted by F^+

There are 6 inference rules that can be used to infer or derive new dependencies from a given set of dependencies.

We use the notation $F \models X \rightarrow Y$ to denote the functional dependency X determines Y is inferred from the set of functional dependencies F.

Inference rules for functional dependency :-

IRule 1 : Reflexive Rule :-

It says that if $Y \subseteq X$ then $X \rightarrow Y$

proof :- Suppose $Y \subseteq X$ then there exists 2 tuples t_1 and t_2 $\in r(R)$ such that $t_1[X] = t_2[X]$ and $t_1[Y] = t_2[Y]$

$\therefore$ According to the defination of functional dependency

$X \rightarrow Y$

trivial FD means : If $Y \subseteq X$ and $X \rightarrow Y$ then called trivial FD

IR 2 :- Augmentation Rule :

if $X \rightarrow Y \models XZ \rightarrow YZ$

Proof :-

assume that $X \rightarrow Y$ holds in a relation instance $r(R)$

$XZ \rightarrow YZ$ does not hold in a relational state $r(R)$. Then there must exists two tuples t_1 and t_2 in R such that $t_1[X] = t_2[X]$ ⟶ ①

$t_1[Y] = t_2[Y]$ ⟶ ②

$t_1[YZ] = t_2[XZ]$ ⟶ ③

$t_1[YZ] \neq t_2[YZ]$ ⟶ ④

this is not possible because ∵ from (1) and (3)

$$t_1[z] = t_2[z] \longrightarrow (5)$$

∵ from (2) and (5)

$$t_1[yz] = t_2[yz] \longrightarrow (6)$$

∵ from (3) and (6)

According to defination of functional dependency

$xz \longrightarrow yz$ which contradicts to our assumption

$\therefore xz \rightarrow yz$

- IR 3 :- Transitive Rule :-

  if $X \longrightarrow Y$ and $Y \longrightarrow Z \models X \longrightarrow Z$

- Proof :- assume that $X \longrightarrow Y$ and $Y \longrightarrow Z$ both hold in a relation state $r(R)$ then there exists any 2 tuples t_1 and t_2

$$t_1[x] = t_2[x] \longrightarrow (1)$$
$$t_1[y] = t_2[y] \longrightarrow (2)$$

and also $t_1[z] = t_2[z] \longrightarrow (3)$

from (1) & (3), according to defn of functional dependency

$X \longrightarrow Z$

IR4: Decomposition or Projective Rule:

If $X \to YZ \models X \to Y, X \to Z$

Proof:- $X \to YZ$ (given) from the reflexive rule we can say that $YZ \to Y$

$YZ \to Z$

∴ according to transitive rule

$X \to Y, X \to Z$

IR5:- Union (or) Additive Rule:-

If $(X \to Y), (X \to Z) \models X \to YZ$

Proof:- Given that $X \to Y$ and $X \to Z$

$X \to Y \to (1)$

$X \to Z \to (2)$

augment X on both sides of (1)

$X \to XY \to (3)$

augment Y on both sides of (2)

$YX \to YZ \to (4)$

∴ according to transitive rule on (3) & (4)

$X \to YZ$

IR6: Pseudo transitive Rule:

if $(X \to Y), (WY \to Z) \models WX \to Z$

proof:- Given $X \to Y, WY \to Z$

$X \to Y \to (1)$

$WY \to Z \to (2)$

augment W on both sides (1)

$WX \to WY \to (3)$

∴ according to transitive (3) & (2)

$WX \to Z$

The first 3 Inference rules

IR1, IR2, IR3 } are called as Armstrong's Inference rules

These are the basic rules all other rules are derived.

(or)

Any functional dependency can be derived from Armstrongs Inference rules.

→ Given a set of functional dependencies F in order to F^+ (closure of F) that is all the functional dependencies that can be inferred from F, we need to determine closure of each set of attributes X i.e X^+, where X appears as a left hand side of some functional dependency in F

Algorithm to find closure of an attribute :-

$X^+ := X$

Repeat

old $X^+ := X^+$

for each functional dependency $Y \rightarrow Z \in F$ do

if $Y \subseteq X^+$, then $X^+ = X^+ \cup Z$

untill ($X^+ = \text{old } X^+$)

Example:- $F = \{$ ssn → Ename, pno → {Pname, Plocation}, {ssn, pno} → hours $\}$

$F^+ = ?$

$Ssn^+ = \{ Ssn, Ename \}$

$Pno^+ = \{ pno, Pname, location \}$

$\{ Ssn, Pno \}^+ = \{ Ssn, Pno, hours, Ename, Pname, Ploc, hours \}$

$F^+ = F \cup \{ \text{New functional dependencies} \}$

$F^+ = F \cup \{ \{ Ssn, Pno \} \rightarrow pno, Pname, Ploc, Ssn, Ename \}$

Equivalence of sets of functional dependencies :-

A set of functional dependencies F is said to cover another set of functional dependencies E, if every functional dependency in E is also in F^+.

That is every dependency in E can be inferred from F, alternatively we say that E is covered by F or F covers E.

Two sets of functional dependencies F and E are said to be equivalent if F covers E and E covers F that is $F^+ = E^+$

Example :- Given a relation R(A, B.....F) and a set of functional dependency $F_1 = \{ A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow F \}$

$F_2 = \{ A \rightarrow CD, E \rightarrow AF \}$

check whether F_1 and F_2 are equivalent ?

Solution :-

i) whether F_1 covers F_2 ?

$F_1 : E \rightarrow AD , E \rightarrow F$

$\Rightarrow E \rightarrow AF$

In F_1

augment A on both side dog

$\rightarrow A \rightarrow C$

$A \rightarrow AC$

$AC \rightarrow D$

$A \rightarrow D$

$\therefore A \rightarrow CD$

$\therefore F_2 \subseteq F^+$

ii) F_2 covers F_1 ?

$A \rightarrow CD$ $\quad A \rightarrow C , A \rightarrow D$ {Decomposition rule}

$E \rightarrow AF$

i) $A \rightarrow C$

$A \rightarrow CD$

$A \rightarrow C \quad A \rightarrow D$

ii) $AC \rightarrow D$

$A \rightarrow CD$

Augment C on both side $\quad AC \rightarrow CD$ } $AC \rightarrow C , AC \rightarrow D$ {decomposition rule}

iii) $E \rightarrow AD$ } Accessing additive rule

$E \rightarrow F$

$E \rightarrow ADF$

3|1|25

according to ~~decomposition rule~~

~~E → AD~~
~~E → F~~

E → AD

E → AF

E → A , E → F

E → CD { by transitive rule }

E → C , E → D

by union rule

✓ E → AD

all functional dependencies of F_1 present in F_2^+

$$F_1^+ = F_2^+$$

$$\therefore F_1 \equiv F_2$$

Minimal sets of functional dependency :- (Minimal covers / Cannonical covers)

Minimal covers of set of functional dependencies E is a set of functional dependencies F that satisfies the property that every dependency in E is in the closure F^+ of F with an additional property that if any dependency from set F is removed thus F will not covers E.

We can formally define a set of functional dependency's F to be minimal if its satisfies the following conditions.

i) Every dependency in F has single attribute for its RHS

ii) We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$ where Y is proper subset of X and

Still have a set of dependencies that is equivalent to F.

iii) We cannot remove any dependency from F and still we have set of dependencies that is equivalent to F.

Algorithm for finding a minimal cover F for a set of functional dependencies E

Step 1: $F = E$

Step 2: Replace each functional dependency $X \rightarrow \{A_1, A_2, \ldots A_n\}$ in F by n functional dependencies

$X \rightarrow A_1$
$X \rightarrow A_2$
$X \rightarrow A_3$
.
.
.
$X \rightarrow A_n$

Step 3: For each functional dependency $X \rightarrow A$ in F, for each attribute B that is an element of X

if $\{F - \{X \rightarrow A\}\} \cup \{\{X - B\} \rightarrow A\}$

equivalent to F then

replace $X \rightarrow A$ with $X - B \rightarrow A$ in F

Step 4: For each remaining functional dependency $X \rightarrow A$ in F,

if $F - \{X \rightarrow A\}$ is equivalent to F then

remove $X \rightarrow A$ from F

Example:

Sen → Ename

Ename → Sal

Sen → Sal

{Sen, Pno} → hours

{Sen, Pno, dno} → hours

Step2 will be split multiple attributes of right hand side

Step3 It will check the redudancy on left hand side

Step4:- It will check redudancy with respect to functional dependency.

Example:

Let R = (A - - - - - - F)

F_1 = { A → C, AC → D, E → AD, E → F }

Find minimal set dos F_1

step1:- A → C, AC → D, E → A, E → D, E → F

step2:- AC → D

A → C
A → AC
AC → D
A → D

} transitive rule

In AC → D C is redundant as A alone can determine D ∴ AC → D can be replaced

A → D

Now A → C, A → D, E → A, E → D, EF

From A → D, E → A we can derive E → D hence we remove E → D.

∴ the minimal covers set of functional dependency covers

$$\left.\begin{array}{l} A \to C \\ A \to D \\ E \to A \\ E \to F \end{array}\right\} \equiv F$$

Compute the closure of the following

→ Find out the key attribute of the relational schema $R = (A \text{ - - - - - } E)$, given the functional dependencies $F = \{A \to BC, CD \to E, B \to D, E \to A\}$.

Sol:-

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

∴ A is key

$CD^+ = \{C, D, E, A, B,\}$

CD is key

$B^+ = \{B, D\}$ B is not key

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

E is also a key

Normalization of relations :-

Normalization of data is a process of analyzing the given relational schema based on their functional dependency and primary key to achieve the following properties:

i] Minimizing redundancy

ii] Minimising insertion, updation, deletion anamolies

iii] loss less join or non additive join property

iv] Dependency preservation property

Different forms

Unsatisfactory relation schemas that do not meet certain conditions or normal form tests are decomposed into smaller relation schemas that meet the normal forms and hence satisfy the above said properties.

Definition :-

prime attribute: An attribute of relation schema R is called as prime attribute of R if it is a member of some candidate key of R, otherwise the attribute is said to be non prime.

First Normal Form:-

First normal form deserves multivalued attributes and nested relations within a relation

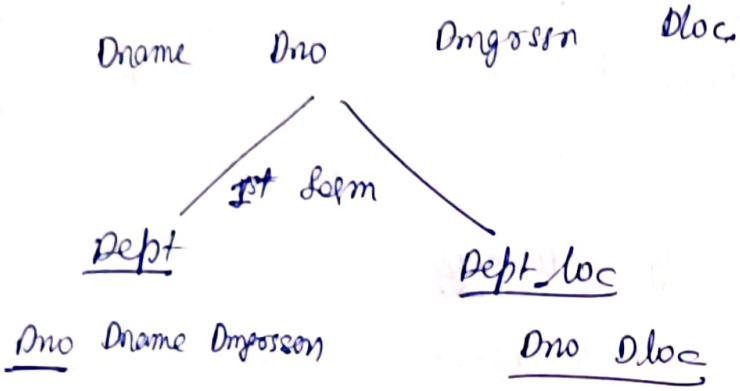
Ex: consider relation R Dept

Dname	Dno	Dmgr_ssn	Dloc

where a dept can have multiple locations

* First normal form not dependent on functional dependency.

considered relation :



Ex2:- Example for nested relations. (~~not~~ first normal form ~~not~~ allow nested relation)

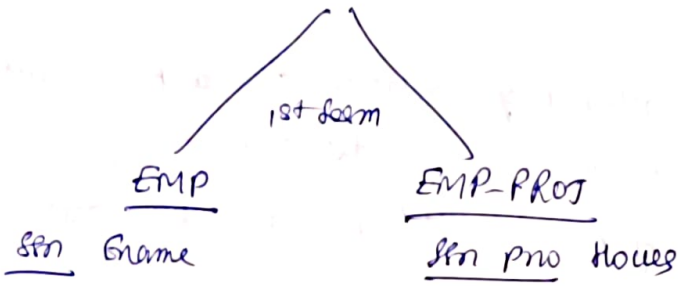
EMP_PROJ

Ssn Sname Pno Hours

where an employee can work on more than one project

EMP-PROJ

Ssn Sname Pno Hours



Second normal Form:-

second normal form is based on the concept of full functional dependency.

A functional dependency $X \rightarrow Y$ is said to be full functional dependency if removal any attribute A from X means that dependency does not hold any more, otherwise the dependency is said to be partial dependency.

Ex:- $Ssn, pno \rightarrow hours$ } full functional dependency

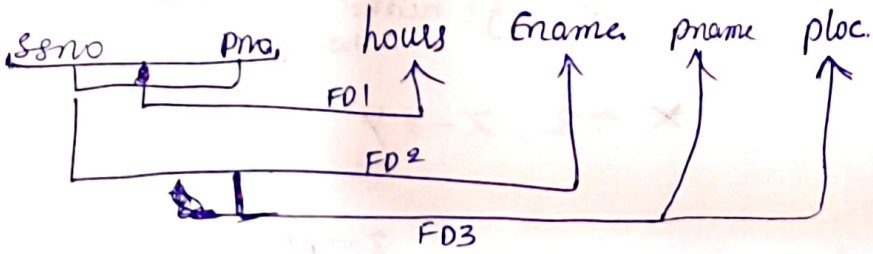
$Ssn \rightarrow name$

$Ssn, Dno \rightarrow name$ } partially functional dependency.

The relational schema R is said to be in second normal form if it satisfies first normal form with an additional property that every non prime attribute A in R is full functionally dependent on any key of R

Example:-

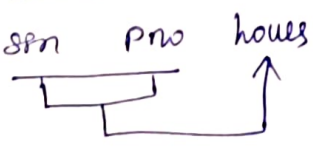
R



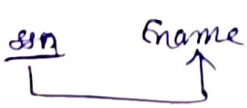
FD2 & FD3 are violates the second normal form.

Now bringing it to normal form by decomposing R

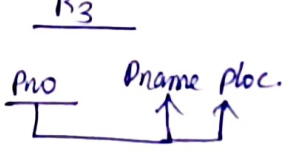
R_1



R_2



R_3

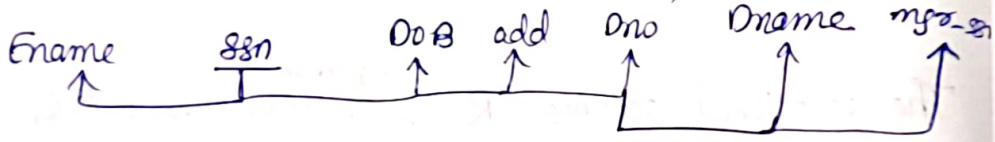


Third Normal Form:-

Third normal form is based on the concept of transitive dependency. A functional dependency $X \rightarrow Y$ in relational schema R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

Definition:- The relation schema R is in third normal form if it satisfies second normal form with an additional property that no non prime attribute of R is transitively dependent on primary key.

Example:-



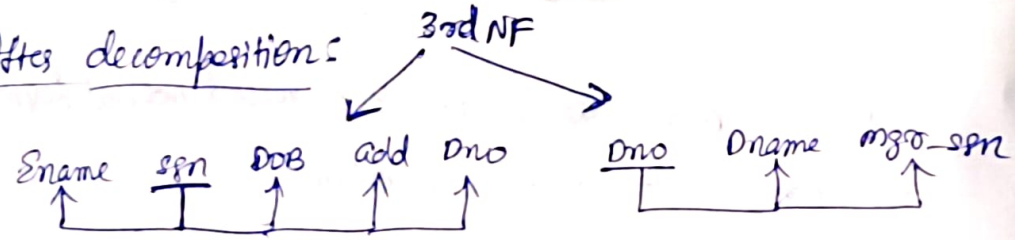
A dependency

$X \rightarrow Y$

Z } neither candidate nor prime

$X \rightarrow Z, Z \rightarrow Y$

After decomposition:-



Another defination of 3NF :-

The relational schema R is in 3NF if whenever a non trivial functional dependency $X \rightarrow A$ holds in R either

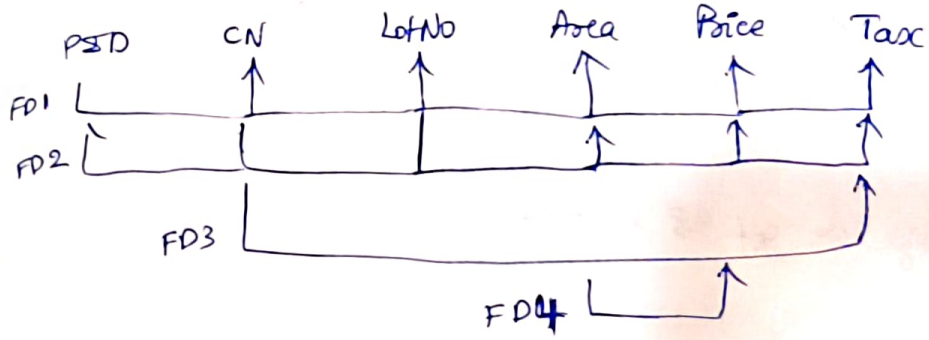
a] X is a superkey of R (OR)

b] A is prime attribute of R

problem :-

Identify which functional dependency violates 2nd, 3rd normal forms and decompose them.

LOTS



solution :-

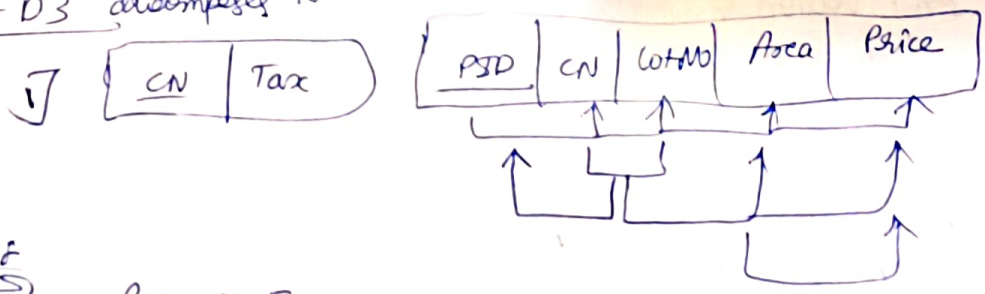
FD1 satisfies 2NF and 3NF

FD2 satisfies both 2NF and 3NF

FD3 violates 2NF

FD4 violates 3NF

FD3 decomposes to 2NF



Note :-

$R \longleftrightarrow F$

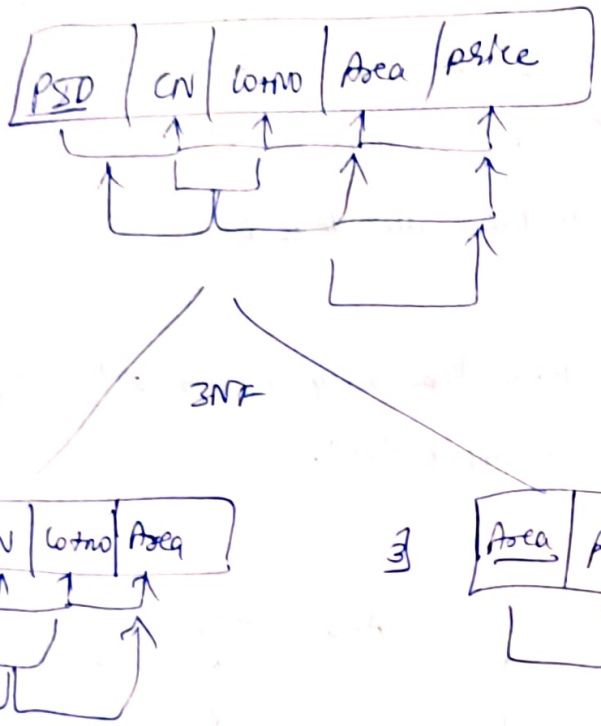
R_1 R_2

F_1 F_2

$F_1^+ \cup F_2^+ = F$

{ dependency preservation property }

Then decomposes to 3NF



After decomposing we got 3 relations.

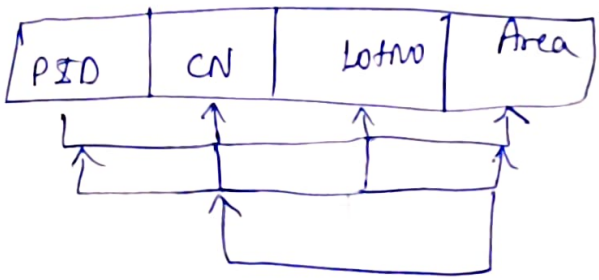
Boyce-codde NF (BCNF) :-

A relation schema R is in BCNF if whenever a non trivial functional dependency $X \longrightarrow A$ codded in R then X is a superkey of R

- It is strict version of 3NF, it avoides partial dependency and transitive dependency

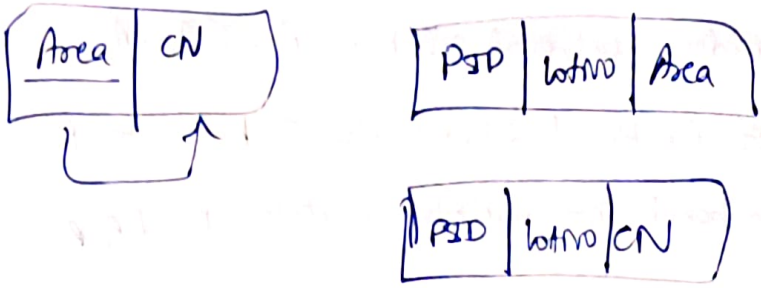
Example :-

LOT



Area → CN violates BCNF because it is not a superkey

decomposed into multiple relation



Whenever BCNF form need to get satisfy when we are decomposing relation we need to compromising the
- loss less join property (It is more preference)
- dependency preservation property

Whenever a relation is decomposed into multiple relations to satisfy BCNF there is a possibility of losing either of functional dependency preservation property or loss less join property, always give preference to loss less join property.

∴ To satisfy BCNF given relation must be decomposed to satisfy lossless join property by compromising functional dependency preservation property.

Properties of relational decomposition :-

i) Attribute preservation property :

Consider a universal relation schema $R = (A_1, A_2 ---- An)$ and a set of functional dependency F that should hold on R.

Let R is decomposed into a set of relation schema's

$D = \{R_1, R_2 ------ Rm\}$ then if $\bigcup_{i=1}^{m} R_i = R$ then

we say that D satisfies attribute preservation property.

ii) Dependency preservation property of a decomposition :

Let us consider a universal relation $R = (A_1, A_2 \dots A_n)$ and a set of functional dependencies F hold on R. Let R is decomposed into multiple relation $D = \{R_1, R_2, \dots R_m\}$ and F_i where $1 \le i \le m$ is the set of functional dependencies hold on R_i if $(F_1 \cup F_2 \cup \dots F_m)^+ = F^+$ then we say that D satisfies functional dependency preservation property.

iii) Non additive or loss less join property :

A decomposition $D = \{R_1, R_2 \dots R_m\}$ of a universal relation R has loss less or non additive join property with respect to a set of dependencies F on R, if for every relation state $r(R)$ that satisfies F.

The following should hold.

i.e., $*(r_1, r_2, \dots r_m)$ should be equal to r

where $*$ is the natural join of all relations in D. ~~and each functional dependency F_i~~ and $R_i = r(R_i)$ for $1 \le i \le m$

Algorithm : for testing non additive join property :-

Input : A universal relation R, A decomposition $D = \{R_1, R_2 \dots R_m\}$ of R and set F of functional dependencies

Step 1 : Create an initial matrix S with one row i, for each relation R_i in D and one column j for each attribute A_j in R.

Step2: Set $S(i,j) = b_{ij}$ for all matrix entries.

Step3: For each row i representing relation schema R_i

for each column j representing attribute A_j

if (relation R_i includes attribute A_j)

then

Set $S(i,j) = a_j$

Step4: Repeat the following loop until a complete loop execution results in no changes to S.

for each functional dependency $X \rightarrow Y$ in F

for all rows in S that have the same symbol in the columns corresponding to attributes in X

Make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows; if any of the rows has an a symbol for the column set the other rows to the same a symbol in the column.

if no 'a' symbol exists for the attribute in any of the rows choose one of the b symbols that appears in one of the rows for the attributes, and set the other rows to the same b symbol in the column

Step5: If a row is made up entirely of 'a' symbols then the decomposition has the non additive join property otherwise it does not.

Example:- (1)

$R = \{ ssn, Ename, Pno, Pname, Ploc, hours \}$

$R_1 = \{ ssn, Ename \}$

$R_2 = \{ Pno, Pname, Ploc \}$

$R_3 = \{ ssn, Pno, hours \}$

$F = \{ ssn \rightarrow Ename, Pno \rightarrow \{Pname, Ploc\}$

$\{ ssn, Pno \} \rightarrow hours \}$

According algorithm:-

	A_1 ssn	A_2 Ename	A_3 Pno	A_4 Pname	A_5 Ploc	A_6 hours
R_1	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}	b_{26}
R_3	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}	b_{36}

	A_1 ssn	A_2 Ename	A_3 Pno	A_4 Pname	A_5 Ploc	A_6 hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

	ssn	Ename	Pno	Pname	Ploc	hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	a_2	a_3	a_4	a_5	a_6

Since the R_3 contains all a symbols and hence the given decomposition satisfies loss-less join property.

(2) $R = \{ ssn, Ename, Pno, Pname, ploc, hours \}$

$R_1 = \{ Ename, ploc \}$

$R_2 = \{ ssn, Pno, hours, Pname, ploc \}$

$F = \{ ssn \rightarrow Ename, Pno \rightarrow \{ Pname, ploc \}, \{ ssn, pno \} \rightarrow hours \}$

According to algorithm

	ssn	Ename	Pno	Pname	Ploc	hours
R_1	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}	b_{26}

	ssn	Ename	Pno	Pname	Ploc	hours
R_1	b_{11}	a_2	b_{13}	b_{14}	~~b_{25}~~ a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

	ssn	Ename	Pno	Pname	Ploc	hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	b_{24} a_4	a_5	a_6

Hence no row contains all a symbol thus decomposition is lossed join.

- Non editive join test for binary decomposition :

A decomposition $D = \{R_1, R_2\}$ of R has the loss less join property with respect to a set of functional dependencies F on R if and only if

i] The functional dependency $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ is in F^+

(or)

$(R_1 \cap R_2) \rightarrow (R_2 - R_1)$ is in F^+.