(1)

UNIT-1

# INTRODUCTION:

## Data:-

Raw facts collected from environment about physical Phenomena or business transactions.

## Database:-

A database is an organized collection of logically related data that contains information relevant to an enterprise. The database is also called the repository or container for a collection of data files. For example :- university database maintains information about students, courses and grades in university.

## Database Management System (DBMS):-

A Database Management System (DBMS) is the set of programs that is used to store, retrieve and manipulate the data in convenient and efficient way. Main goal of DBMS is to hide underlying complexities of data management from users and provide easy interface to them. Some common example of DBMS software are Oracle, MySQL(mostly used), Sybase, DB2, Microsoft SQL server, Ms-Access, SQLight, etc.

DBMS that maintains relationship between multiple data files is called relational database management system (RDBMS).

(2)

A database system consists of database, DBMS and application programs. For example :- In a DBMS, the library, accounting management system and examination system program could have a common database. This database approach to data processing is as shown in figure below:

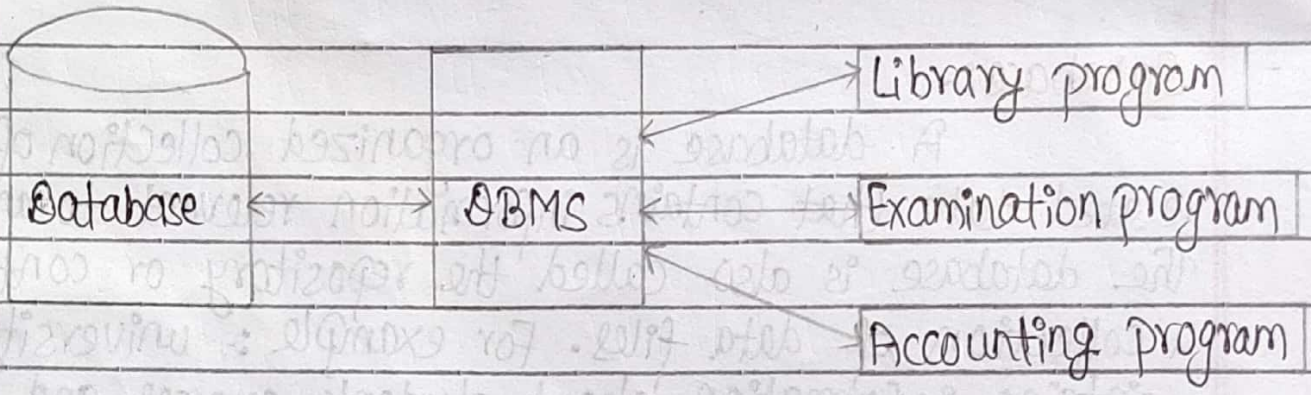



Fig:- Database System

## Application of Database System:-

Database Systems are widely used in different areas because of their numerous advantages. Some of the most common database applications are listed here:

1) Banking:-

for customer information, accounts and loans and banking transactions.

ii) Airlines and Railways:-

Airlines and Railways use online databases for reservation and for displaying the schedule information.

iii) Universities:-

for student information, course registration and grades information.

iv) Telecommunication:-

Telecommunication departments use database to store information about the communication network, telephone numbers, record of calls, for generating monthly bills, etc.

v) Credit card transactions:-

Databases are used for keeping track of purchases on credit cards inorder to generate monthly statements.

vi) Finance:-

Databases are used for storing information such as sales, purchases of stocks and bonds or data useful for online trading.

vii) Sales:-

Databases are used to store product, customer and transaction details.

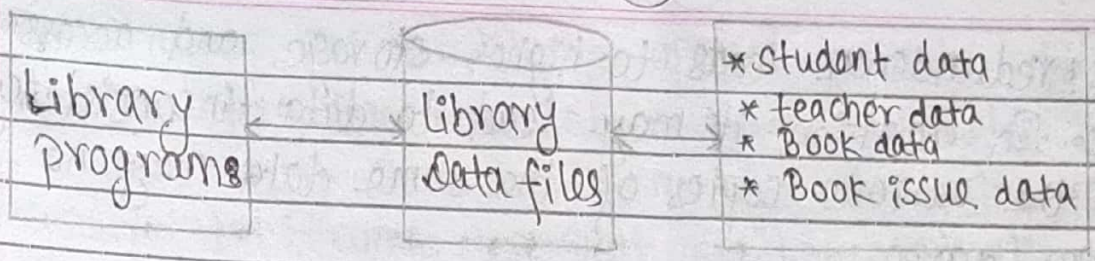
VIII) Human Resources:-

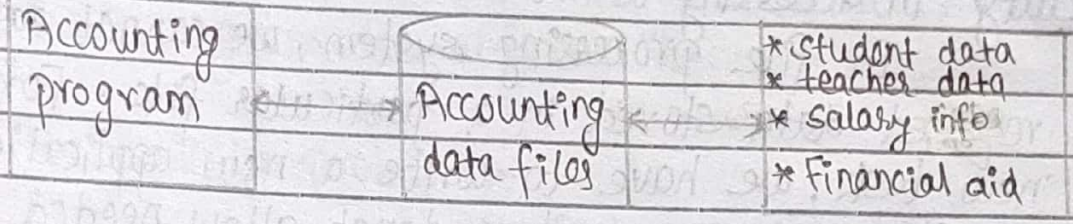
For information about employees, salaries, payroll taxes & benefits and for generation of pay checks.

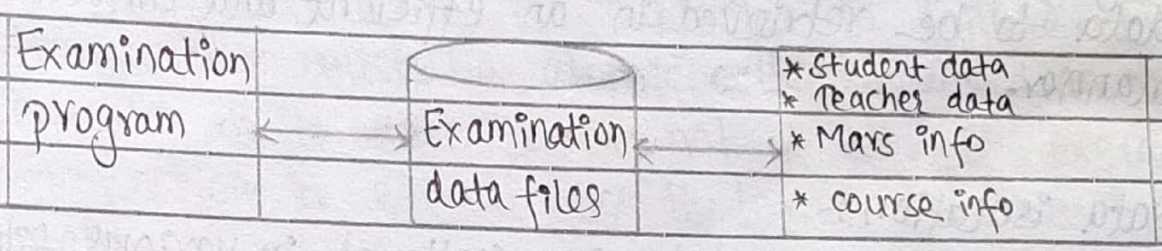
File Management System (FMS):-

File Management Systems are also called flat file systems. It stores data in a plain text file. A flat file is a file that contains records in which each record is specified in a single line. Fields from each record may be simply have a fixed width with padding, or may be delimited by white space, tabs, commas or other characters. Extra formatting may be needed to avoid delimiter collision. There are number of structural relationship and the data are flat as in a sheet of papers.

In this approach, each application has data files related to it, containing all the data records needed by the application. Thus an organization has to develop no. of application program, each with an associated application specific data files. For example in a college MIS the library programs, accounting programs would have their own data files as shown in figure below:




## Limitation of flat file system:-

keeping organizational information in a file processing system has a number of major disadvantages. These drawbacks of flat file system are resolved by database management system. Drawbacks of file processing system or advantages of DBMS are described below:

### 1) Data Redundency and inconsistency:-

Data redundency means duplication of same data or data files in different places. Flat file systems are suffered from the problem of high data redundency.

This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency i.e. the various copies of the same data may no longer agree.

II) **Difficulty in accessing data:-**

In file processing system, we cannot easily access required data stored in a particular file. For each new task, we have to write a new application program. File processing system donot allow needed data to be retrieved in an efficient and convenient manner.

III) **Data isolation:-**

Because data are scattered in various files and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult in flat file system. DBMS provide shared access to centrally stored data therefore it is easy for application programs to retrieve required data from centralized database.

IV) **Integrity problem:-**

Integrity means correctness of data before and after execution of a transaction. The data values stored in the database must satisfy certain types of integrity constraints. Integrity

constraints are conditions applied to data. For example; the balance of a bank account must never fall below a prescribed amount (say Rs.500). In a file processing system, integrity constraints become the part of the application program so programmer need to write appropriate code to enforce it. When new constraints are required, it is difficult to change the program to enforce it.

v) Automicity problem :-

A computer system like any other mechanical or electrical device is subjected to failure. Execution of transaction must be atomic. This means transaction must execute at its entirity or not at all. If execution of transaction is not atomic, it leaves database in incorrect state.

vi) Concurrent access anomalies :-

To increase the overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment interaction of concurrent update may result inconsistent data.

Flat file system do not support execution of concurrent transactions and hence may suffered from the problem of inconsistency of data but DBMS support concurrent execution of transactions on the same data without resulting into inconsistent data.

vii) Security problem:-

Not every user of the database system should be able to access all the data. In database system, we may create different user accounts and provide different authentication to different users. Thus, we are able to hide certain information from some users since file processing system consists of number of application programs so it is difficult to enforce security to each application to allow accessing only part of data for individual users.

## Advantages of file system:

Despite of the number of drawbacks of flat file system, they are beneficial in many situations and DBMS may not be suitable in such situations. Benefits of file processing system or disadvantages of DBMS are as follows:

1) Initial Investment:-

To use the database system, we need to purchase DBMS based on our requirement. Besides this, we need to purchase a powerful computer to run it as database server. Due to this reason, initial investment is high if we use DBMS. But such a high initial investment is not needed for flat file system. It can be executed in existing

infrastructure or we may need to purchase general purpose desktop computers.

ii) Dedicated staff :-

To manage database system efficiently, we need to hire a skilled dedicated staff called database administrator (DBA). File processing system do not need to hire such dedicated staff.

iii) Overhead :-

We need to update and maintain database systems in periodic bases to make it efficient in new technologies which is not needed in flat file system.

When flat-file systems are suitable :-

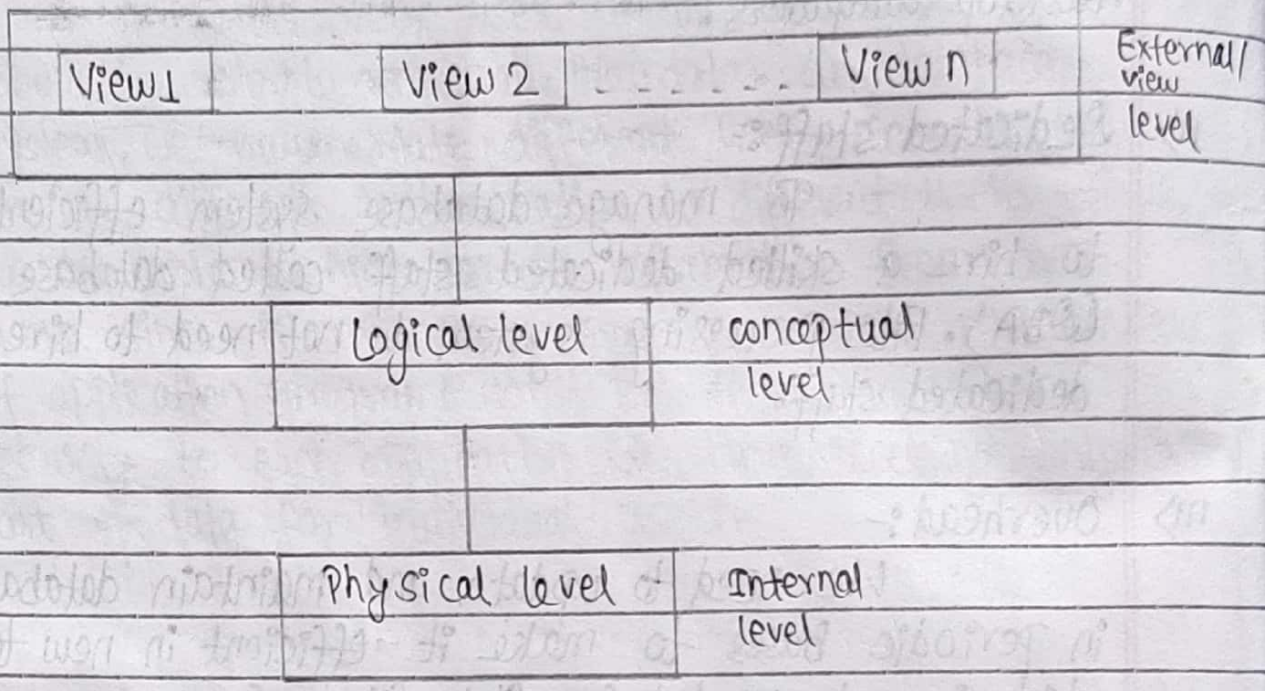
Flat-file systems are more suitable to use rather than DBMS in following situations:

- When system to be developed is simple and small.
- When we need to manage few data.
- When security is not major concern.
- When concurrent access is not needed.

# (3-level architecture or 3-schema architecture or, ANSI/SPARC architecture)

(needed for data independence)

## Data Abstraction :-




[Fig :- Three level of data abstraction]

Data abstraction is a technique of hiding the complexity of the database from its users. There are three levels of data abstraction which are discussed below:

1) **Physical level or Internal level:**

It is the lowest level of abstraction and describes how the data in the database are actually stored. The physical level describes complex low level data structures in detail. The data only exists at physical level.

ii) Logical or Conceptual level:

This is the next higher level of abstraction and describes what data are stored in the database and what relationship exist among those data. It describes the structure of the whole database and hides details of physical storage structure. Although implementation of the simple structure at the logical level may involve complex physical level structures, the user of the logical level doesnot need to be aware of this complexity.

iii) View level or External level:

It is the highest level of abstraction and is concerned with the way the data is seen by individual users. This level simplifies the user's interaction with the system. It includes a number of views and database users see only those views.

It describes only those part of database in which the user is interested and hides the rest of the all from those users. Schema at this level is called external schema.

Figure above shows the relationship among these three level of abstraction.

Example :-

⇒ view level
* view result
* view student information

⇒ logical level : entire database schema
* courses (courseNo, CourseName, credits, Dept)
* Student (StudentId, StudentName, Level, Major)
* Grade (StudentID, CourseNo, Marks)

⇒ Physical level:
* how these tables are stored
* how many bytes it required, etc.

## Data Independence :-

The capacity to change the database schema without affecting the application program is called data independence. Database system provides two types of data independence:

i) Logical data independence
ii) Physical data independence.

(Program logic independence)

### i) Logical data independence :-

The capacity to change conceptual (logical level) schema without having to change associated application programs is called logical data

independence.

The correspondance between a particular external view and the conceptual view is called external conceptual mapping. For example: field name can be changed, several conceptual field name can be combined into a single external field and so on. When modification is done to the conceptual schema, only the external/conceptual mapping need to be changed, if the DBMS fully supports the concept of data independence.

(Program data independence)

ii) Physical data independence :-

The capacity to change the internal schema without affecting application programs is called physical data independence.

The correspondance between the conceptual view and the stored database is called conceptual/internal mapping. It specifies how conceptual records and relations are represented at the internal level. If a change is made to the storage structure definition, then the conceptual/internal mapping must be changed accordingly.

For example: if we can change physical level storage detail such as file structure, indexes as long as conceptual schema remains same without altering associated application programs.

## Mapping :-

The DBMS must transform a request specified on an external schema into a request against the conceptual schema and then into a request on the internal schema for processing over the database. The process of transforming requests and results between ~~labels~~ levels is called mapping.

## Instance and Schema :-

### Instance :-

Database changes over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database. It is also called as database state.

### Schema :-

The overall design of the database is called the database schema. The schema of the database does not change frequently. There are 3 schema, partition according to the level of abstraction.

- The physical schema describes the database design at the physical level.

- The logical schema describes the design of the database at the logical level.

- The schema at the view level is sometimes called sub-schema and describes the view of the database. A database may have several sub-schema or view.

## Data Models :-

The data model is the basic structure or design of the database. It consists of a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical and view levels. Different types of data models are as follows:

### 1) Entity-Relationship model:-

The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects. An entity is a 'thing' or 'object' in the real world that is distinguishable from other objects.

### 2) Relational Model:-

The relational model uses a collection of tables to represent both data and the relationship among those data. Each table has multiple columns and each column has a unique name. Tables are also known as relations. The relational model is an example of record based model.

3) Object-Oriented Model:-

In object-oriented model, data is represented in the form of objects. An object contains values stored in instance variable within the objects. Apart from values, an object contains bodies of code called methods or function. This process of binding data & methods together is known as encapsulation.

Object that contains the same types of values and same methods are grouped into class.

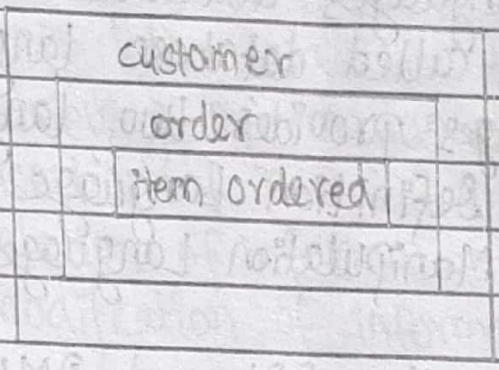
4) Object-Relational Model:-

The object-relational model is a relatively recent development and one in which a lot of research is going nowadays. Object-relational database management systems (ORDBMS) add new object storage capabilities to the relational systems at the core of modern information systems. These new feature integrates management of traditional fielded data, complex objects, such as time series and geospatial data and diverse binary media such as audio, video, images, etc.

5) Hierarchical Database Model:-

The hierarchical data model is the oldest model of database, developed by IBM in 1968. It is the record based representation or implementation of data model. For example; data on customer, order and

items ordered in hierarchy as follows:




To store or retrieve data, the database system must start from the top (also called root).

6) Network Database Model:-

In network database model, network of connection between data elements are established. This model overcomes the problems faced in hierarchical model of searching data from the bottom or middle. The h/w model of previous example discussed in hierarchical model is as follows:

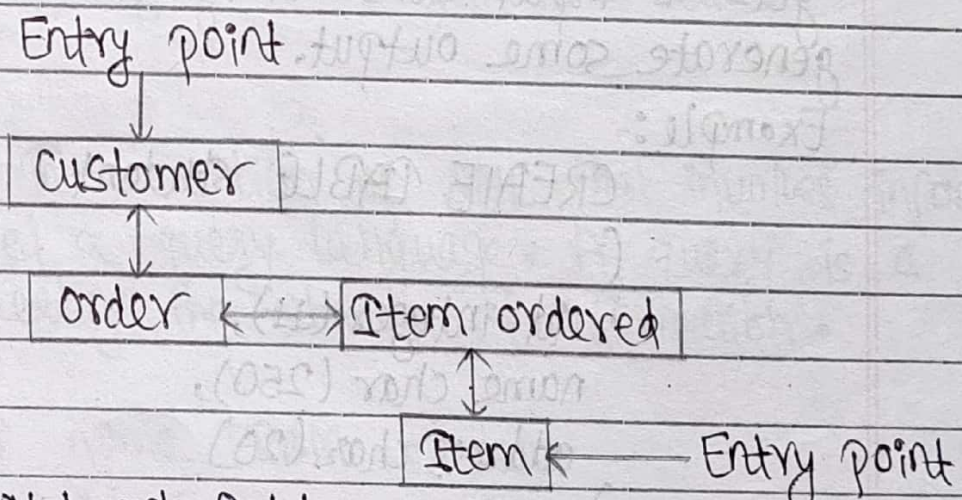



Fig:- Network Database Model

## Database Languages :-

Languages that are used to interact with DBMS are called database languages. The database languages provides two languages :

i) Data Definition Language (DDL)
ii) Data Manipulation Language (DML)

In practice, the DDL and DML are not two separate languages.

### 1) Data Definition Language (DDL) :-

Database language that is used to create, delete or modify database schema is called data definition language (DDL). Its statement are converted into equivalent low level statements by DDL interpreter. DDL is used by Database Administrator or Database designers to specify the conceptual schema of a database. Just like any other programming languages, the DDL gets as input some instructions (statements) and generate some output.

Create
alter
drop

Example :-

```
CREATE TABLE student
(
  id integer (11),
  name char (250),
  address char (20)
);
```

## ii) Data Manipulation Language (DML):-

It is a language that enables users to access or manipulate data i.e. DML is responsible for:

- → retrieving information stored in the database
- → insertion of new information into the database
- → deletion of information from the database
- → modification of information stored in the database.

There are basically two types of DML:-

a) Procedural DML
b) Declarative DML.

### a) Procedural DML:-

Requires a user to specify what data are needed and how to get those data.

### b) Declarative DML (Non-procedural DML):-

Requires a user to specify what data are needed without specifying how to get those data.

The portion of DML that involves information retrieval is called a query language. A query is a statement requesting the retrieval of information.

Example:-

```
SELECT name FROM student
WHERE id=5;
```

## Data Storage and Querying (Storage Manager & Query Processor) :-

A database system is partitioned into modules that deals with different responsibilities of the system. The functional components of a database system can be broadly divided into two components:

- storage manager
- Query Processor

### Storage Manager :-

It is a component of a database system that provides the interface between the low-level data stored in the database and application programs and queries submitted to the system. It is responsible for the interaction with the file manager. It translates the various DML statements into low-level file system commands. Thus, it is responsible for storing, retrieving and updating in the database.

The components of storage manager are:

a) Authorization & integrity manager :-

It is responsible to test for satisfication of integrity contraints and checks the authority of users to access data.

b) Transaction Manager:-

It ensures that the database remains in a consistent state despite system failure and that concurrent transaction execution proceeds without conflicting.

c) File Manager:-

It manages the allocation of the space on disk storage and the data structures used to represent information stored on disk.

d) Buffer Manager:-

It is responsible for fetching data from disk storage into main memory and deciding what data to cache in main memory

Query Processor:-

It helps the database system to simplify and facilitate to access data. It allows database users to obtain good performance while working at view level. and does not need to understand the physical level details of the implementation of the system. The components of query processor includes:

a) DDL Interpreter:-

It interpretes DDL statements and record definitions in the data dictionary.

b) DML compiler :-

It translates DML statements in a query language in an evaluation plan consisting of low-level instructions that the query evaluation engine understand.

c) Query Evaluation Engine :-

It executes low-level instructions generated by DML compiler.

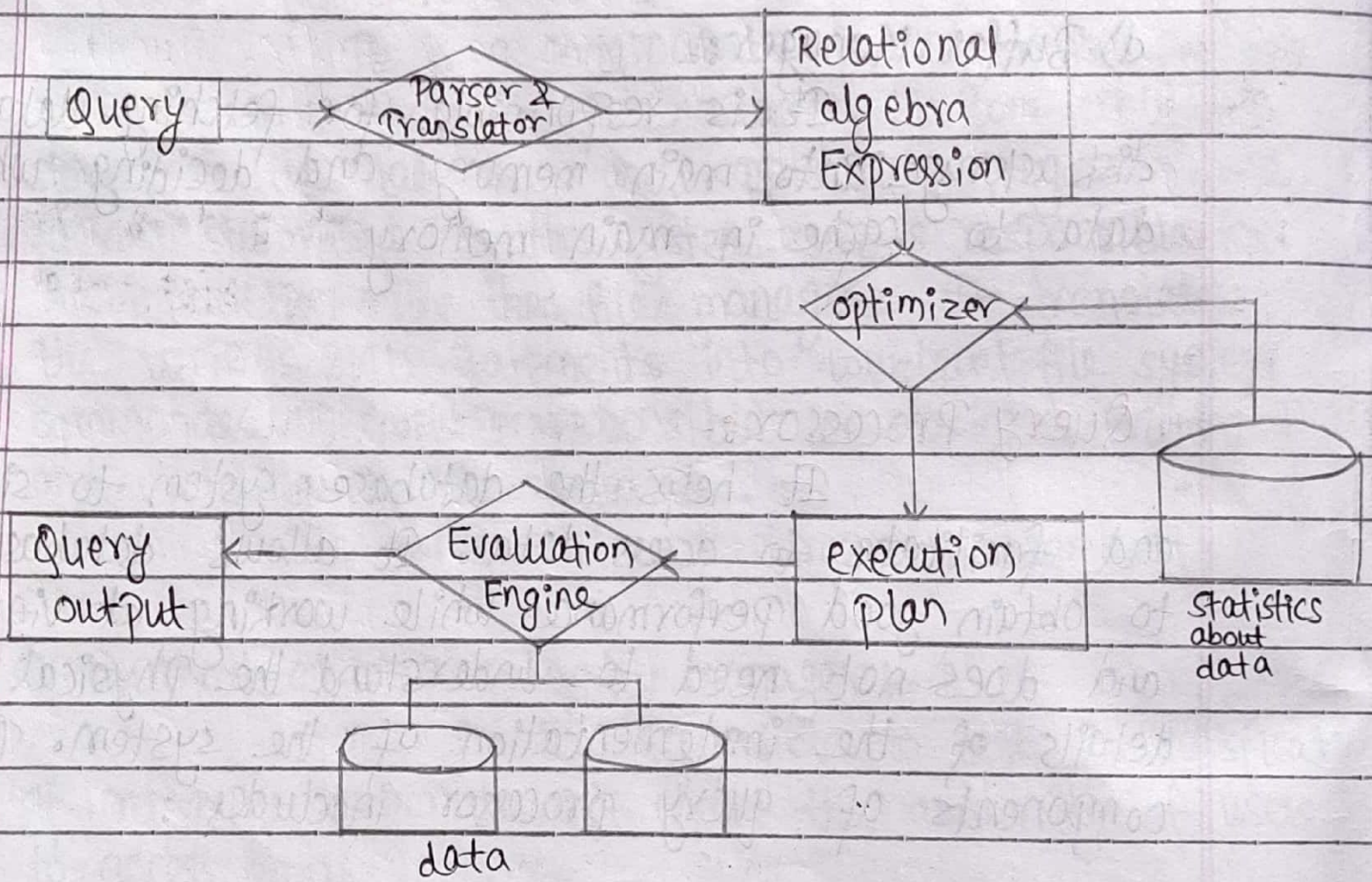



Fig :- Query Processing step

## Database Users and Administrators:

A primary goal of a database system is to retrieve information from and store new information into the database. People who work with a database can be categorized as database users and database administrators.

### Database Users:-

Database users are those who interact with database in order to query and update the database, and generate reports. On the basis of way of interaction with database system, database users are classified into following categories:

1) Naive User — interact with database by already used apps.
2) Application Programmer — write application for naive user
3) Sophisticated user
4) Specialized user

### 1) Naive Users:-

The users who query and update the database by invoking some already written application programs. For example: the owner of the book store enters the details of various books in the database by invoking appropriate application program. The naive user interact with the database using form interface.

2) Application programmer:-
They are computer professionals who write application programs. They can choose from many tools to develop user interfaces. The database application programmer develops application program to facilitate easy access for the database users.

Command line bata user sanga interact garxa

3) Sophisticated users:-
These users interact with the system without writing programs. Instead, they form their request either using a database query language or by using tools such as data analysis software. Analysts who submit query to database fall in this category.

4) Specialized users:-
Specialized users are sophisticated users who write specialized database application that do not fit into the traditional data processing system. Computer Aided Design systems, knowledge based and expert systems are examples of these applications.

## Database Administrators (DBA) :-

DBA is a person who has central control over both data and application programs. The role of DBA includes:

i) Schema Definition and Modification
ii) Storage structure & access method definition
iii) Physical organization modification
iv) Granting of authorization of data access
v) Routine Maintenance

### 1) Schema Definition and Modification:-

The DBA creates the database schema by executing a set of data definition statements in the DDL. The DBA also carries out the changes to the schema according to the changing needs of an organization.

### 2) Storage structure and access method definition:-

Responsible for defining how actually data are stored and how they get.

### 3) Physical organization modification:-

The DBA carries out changes to physical organization to reflect the changing needs of organization or to alter the physical organization to improve performance.

### 4) Granting of authorization of data access:-

By generating different types of authorization, the DBA can regulate which parts of the database

various users can access. The authorization information is kept in a special system structure that the database system consult whenever someone attempts to access the data in the system.

5) Routine-Maintenance :-

Examples of database administrator's routine-maintenance activities are:

- Periodically backing up the database either onto tapes or onto remote servers to prevent loss of data incase of disasters.
- Ensuring that enough free disk space is available for normal operations and upgrading disk space as required.
- Monitoring jobs running on the database and ensuring that performance is not degraded by every expensive task submitted by some users.

## Data Administration and Database Administration:

Many organization do not recognize the essential difference between data administration and database administration. Due to this, there exist confusion in responsibility. For proper management of the corporate resources of information, these activities never be combined in a person or sub-group.

Following are the major responsibilities of data administrator and database administrator:

1) Data Administration (Logical Design):-
- Perform business requirement gathering
- Analyze requirement
- Model business based on requirement
- Define an enforce standards and conversions
- Conduct data definition sessions with users.
- Assist database administration in creating physical tables from logical model.

2) Database Administration (Physical / Operation Design):-
- Define requirement parameters for database definition
- Analyze data volumes and space requirements.
- Executes database back-ups and recoveries
- Monitor database space requirement
- Verify integrity of data in database
- Co-ordinate the transforming of logical structure to properly performing physical structure.

## DBMS Interface :-

A DBMS interface is the abstraction of a piece of functionality of a DBMS. It usually refers to the communication boundary between the DBMS and clients or the abstraction provided by a component within a DBMS. A DBMS interface hides the implementation of functionality of the DBMS. Following are the different types of DBMS interfaces :

1) Menu-based interface for web clients :-
These interface present user with list of options called menus that lead the user through the formulation of request.

11) Form-based interface :-
A form-based interface display a form to the user. User can fill out all of the form entries to insert new data or they fill out certain entries in which case the DBMS will retrieve matching data for remaining entries.

111) Graphical User Interface (GUI) :-
A GUI typically displays schema to the user in diagrammatic form. The user can then specify a query by manipulating the diagram.

Date
Page

iv) Natural Language Interface :-

These language accept request written in English or some other languages. A natural language interface usually has its own schema which is similar to the database conceptual schema.

(enter function (ie repeatedly use same user → parametric users)

v) Interface for parametric users :-

Parametric users such as bank tellers often have a small set of operations that they must perform repeatedly. Usually, a small set of abbreviated commands is included with the goal of minimizing the no. of key strokes required for each request.

vi) Interface for DBA :-

Most database system contains previleged command that can be used only by DBA staff. These includes commands for creating accounts, setting system parameters, granting accounts authorizations, etc.

Communication Manager :-

The data communication manager is a software component that manages all the message transmission between the user and the DBMS. More formally between user and some applications running on top of the DBMS.

## Database System Utilities:

In addition to processing the software modules, the most DBMS have database utilities that help the DBA in managing the database system. Common utilities have following type of functions:

1) **Loading utility:-**
A loading utility is used to load existing data file such as text files or sequential files into the database. Usually, the source format of the data file and the target database file structure are specified into the utility which then automatically re-formats data and store it in the database.

2) **Backup:-**
A backup utility creates a backup copy of the database usually by dumping the entire database into tape. The backup copy can be used to restore the database in case of any type of failure.

3) **File organization:-**
This utility can be used to re-organize a database file into a different file organization to improve the performance.

4) Performance monitoring:-

This utility monitors database usage and provide statistics to the DBA. The DBA uses the statistis in making decisions such as whetheter or not to reorganize files to improve the performance.

Database Application Architecture:-

The architecture of a database system is greatly influenced by the underline computer system or on which the database system runs. The database system is divided into of two types:

1) Centralized Architecture:-

In centralized database system, the database system, application programs and user interface all are executed on a single system and dummy terminals are connected to it. The processing power of single system is utilized and dummy terminals are used only to display the information.

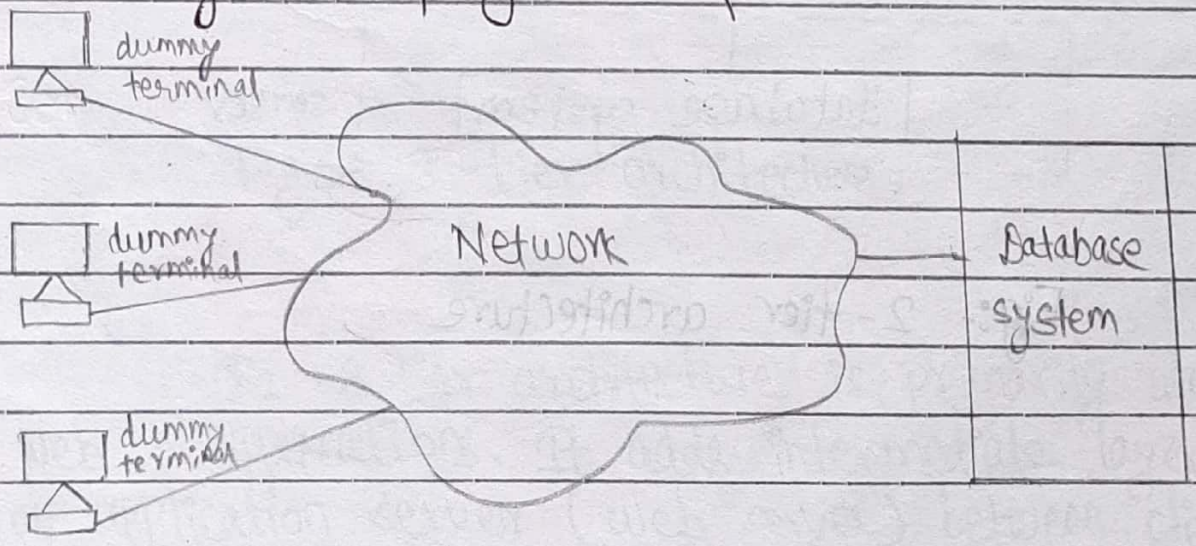



Fig:- Centralized database system

## 2) Client / Server Database System:-

In client-server architecture, the processing power of computer system at the user's end is utilized by processing the user interface on that system. A client is a computer system that sends request to the server connected to the network and a server is a computer system that receives the request, process it and returns the requested information back to the client. There are two approaches to implement the client-server architecture:

### a) Two-tier Architecture:-

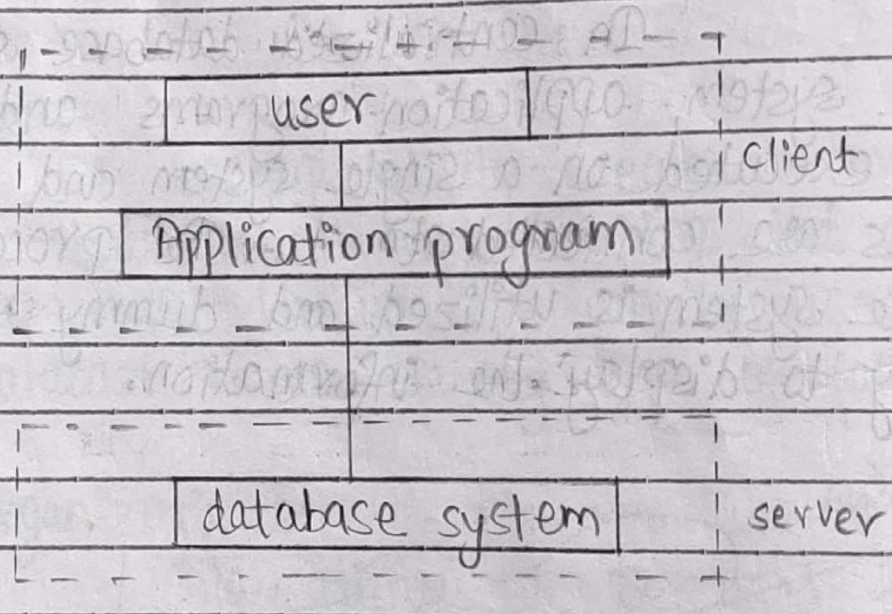



Fig:- 2-tier architecture

In a two-tier architecture, the application resides at the client machine where it invokes database system functionality at the server machine through the query language statements. Application interface standard like ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity) are used for interface between the client and the server

b) Three-tier Architecture:-

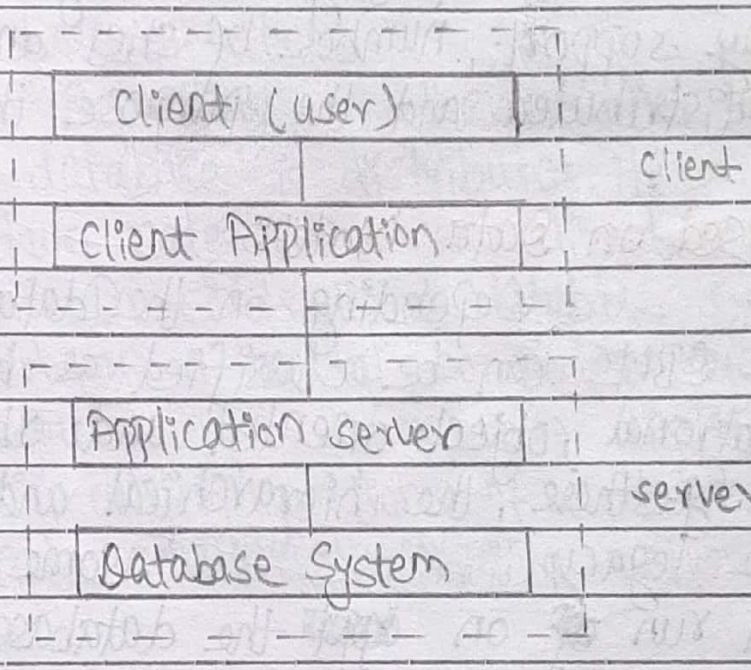



Fig:- 3-tier architecture.

The 3 tier architecture is primarily used for web applications. It adds intermediate layer known as application server (web server) between client and

database server. The client communicates with application server which inturn communicates with database server. It checks client's credential before forwarding a database request to the database server. Hence, it improves database security.

## Classification of OBMS :-

The OBMS can be classified into different categories on the basis of several criteria such as the data model they are using, number of users they support, number of sites on which the database is distributed and the purpose they serve.

### I) Based on Data Model :-

Depending on the data model they use, the OBMS can be classified as hierarchical, network, relational, object-oriented and object-relational. Among these, the hierarchical and network models are legacy (replace garna thaicha but totally replace nabhako). Some of the old applications still run on the database system based on these models. Most of the popular and current commercial OBMS are based on relational data models. The object-oriented data models have been implemented in some OBMS but it is not popular. Due to the popularity of relational database, the object-oriented concept have been introduced in these databases that led to the development of a new class of OBMS

called object relational DBMS.

2) Based on number of users:-

Depending on the number of users the DBMS support, it is divided into two types: single user system and multi user system. In single user system, the DBMS resides on one computer and is only accessed by one user at a time. In multi-user system, multiple user can access the database simultaneously.

3) Based on number of sites:-

Depending on the number of sites over which the database is distributed, it can be classified into two categories: centralized and distributed database system. In centralized database system, all data are placed on a central computer such as Mainframe computer or server. A distributed database is a database in which portions of a database are stored on multiple computers within a network.

4) Based on purpose:-

Depending on the purpose of the DBMS, it can be classified as general purpose and specific purpose. General purpose DBMS aim to meet the needs of as many applications as possible. Specific purpose DBMS is developed to accomplish specific task for example; e-mail database.

# Database System Structure:

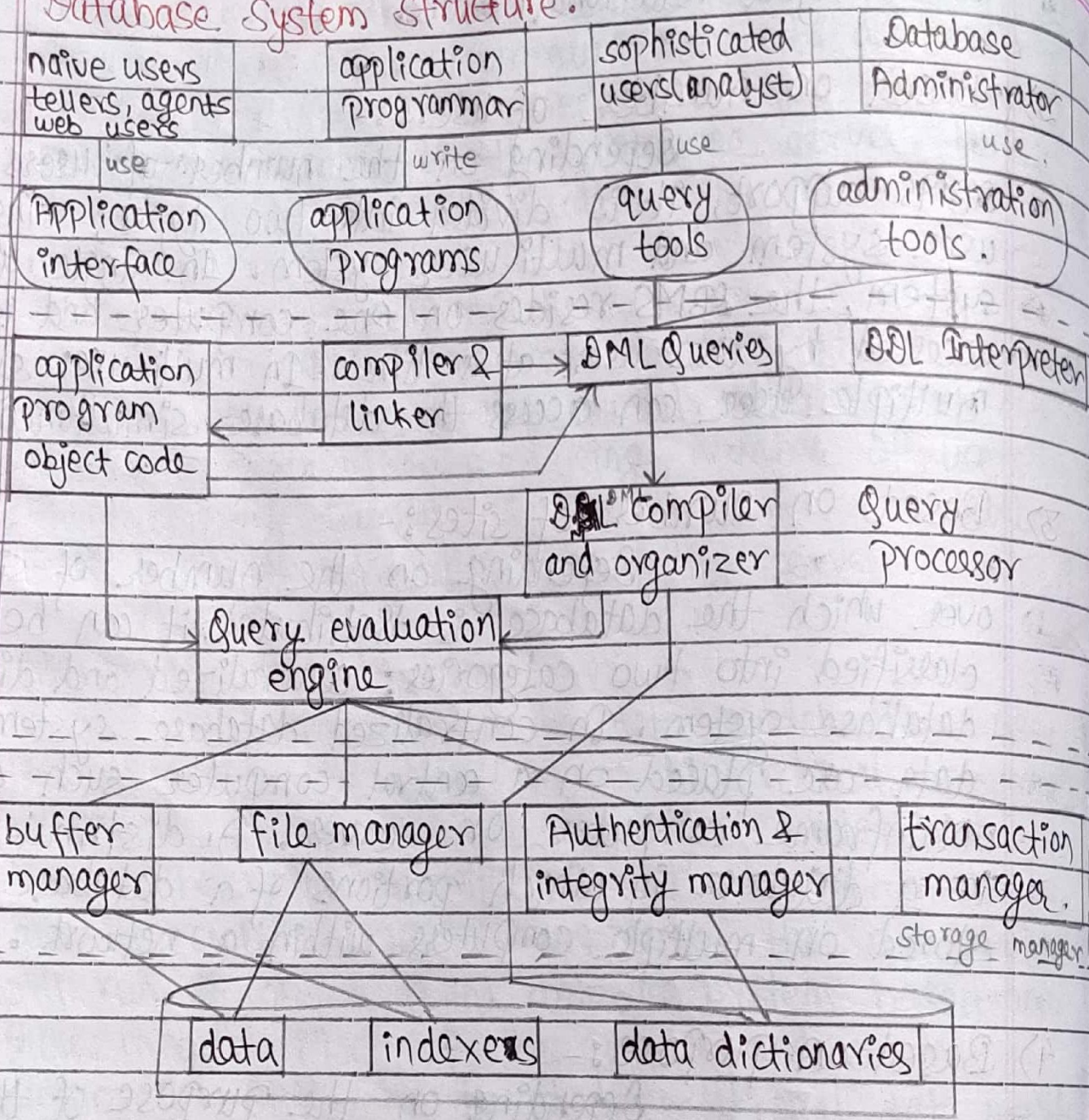



Fig:- Database System Structure.

A database system is partitioned into modules that deals with each of the responsibilities of the overall system. The functional component of database system can be broadly divide into two groups: query processor and storage manager components.

The query processor is important because it helps the database system simplify and facilitate access to data. The storage manager is important because database system typically require large amount of storage space.

1.2 Entity-Relationship Model (abstraction of the system)

## Entity-Relationship Model :-

The entity relationship model is the most popular conceptual model used for designing a database. The ER-model views real world as a set of basic objects called entities, their characteristics called attributes and association among these objects called relationships.

## E-R Diagram :-

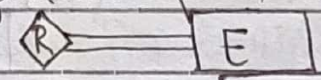
An E-R diagram is a specialized graphical tool that demonstrates the relationship among various entities of a database. It is used to represent overall logical structure of the database. While designing E-R diagrams, the emphasis is on the schema of the database and not on the instances. This is because the schema of the database is changing rarely. However, the instances in the entity-relationship set change frequently.

E-R diagram focus high-level database design and hides low level details of database representation. Therefore, it can be used to communicate with users of the system while collecting information.

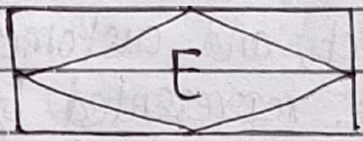
Date
Page

## Symbols used in E-R diagram:

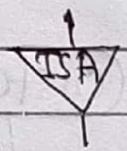
1) Rectangle - represents entity sets.
2) Diamond - represents relationship sets.
3) Lines - links attributes to entity sets and entity sets to relationship sets.
4) Ellipse - represents attributes
5) Double ellipse - represents multivalued attributes
6) Dashed ellipse - represents derived attributes
7) Underline - indicates primary key attributes.
8) Double lines - indicates total participation of an entity set in relationship set.



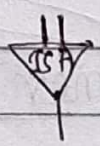

9) Double Rectangle - represents weak entity set.
10) Double Diamond - represent identifying relationship set for weak entity set.
11) Associative entity -




12) Discriminating attribute of weak entity set. -
13) Special / Generalization -




completely separate garna sakine
14) Total Generalization -



# Components of ER model:-

## 1) Entity and Entity Sets:-

An entity is a thing or object in the real world that is distinguishable from another object. For eg:- specific person, company, event, plant, etc.

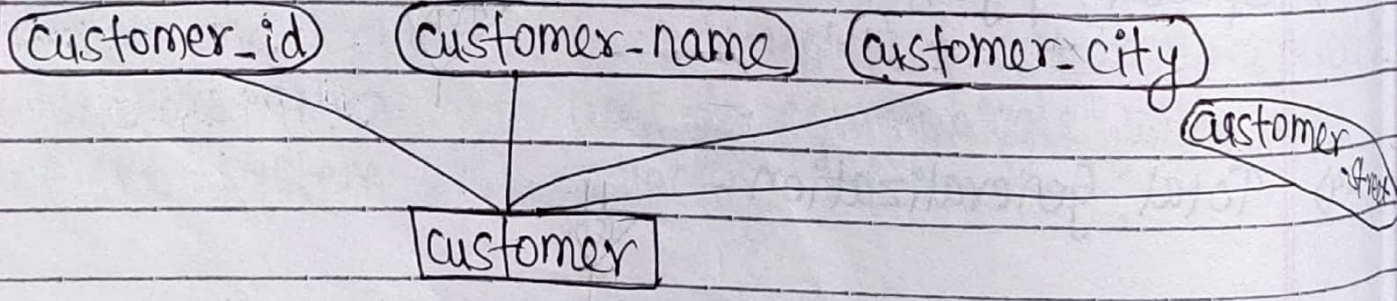
An entity set is a collection of entities of the same type that share the same properties. It is represented by means of rectangles.

For example:

[student], [Teacher], [Projects]

## 2) Attribute:-

Attributes are the information that explains the properties of an entity. For example; a customer entity can have; customer_id, customer_name, customer_street, and customer_city as attributes. Attributes are represented by means of ellipse. Every ellipse represents one attribute and is directly connected to its entity.



For each attribute, there is a set of permitted values called the domain or values set of that attributes. For example: set of all strings of certain length for customer_name is the domain of that attribute.

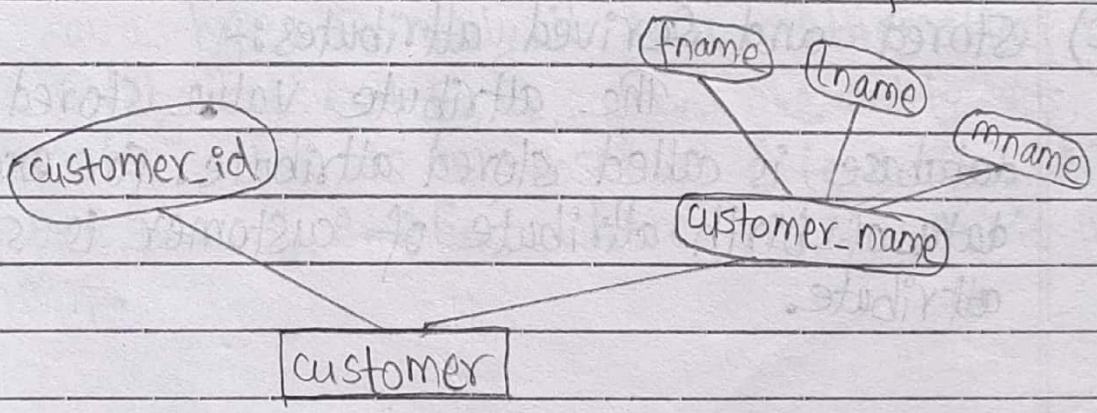
## Types of attributes:

An attribute can be characterized by following attribute types:

a) Simple and composite attributes:-

Attributes which cannot be divided into sub-parts are called simple attributes. For example:- customer_id is simple attribute in customer entity. It cannot be divided into sub-parts.

The attributes that can be divided into sub-parts are called composite attributes. For example:- customer_name in customer entity set is composite attribute since it can be divided into sub parts: fname, mname and lname. Composite attributes are represented by ellipse that are connected with an ellipse



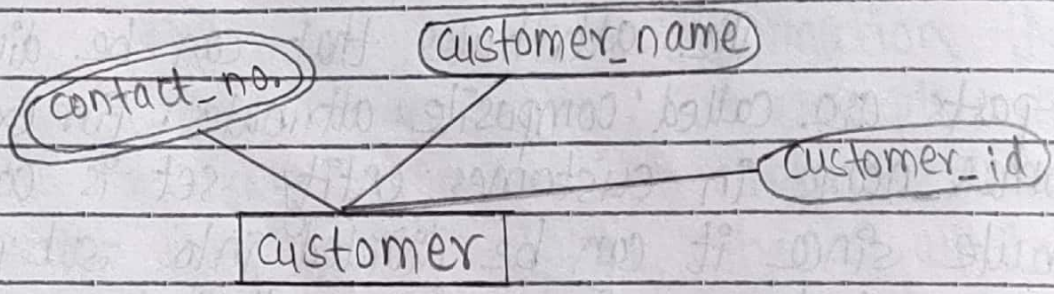
b) Single valued and multivalued attributes:-

The attribute that can take only one value in every entry is called single-valued attribute.

- For examples; attribute customer_name in customer entity set is single-valued attribute since it cannot contain more than one customer_name in any entry.

The attribute that can take more than one values in any entry is called multi-valued attribute.

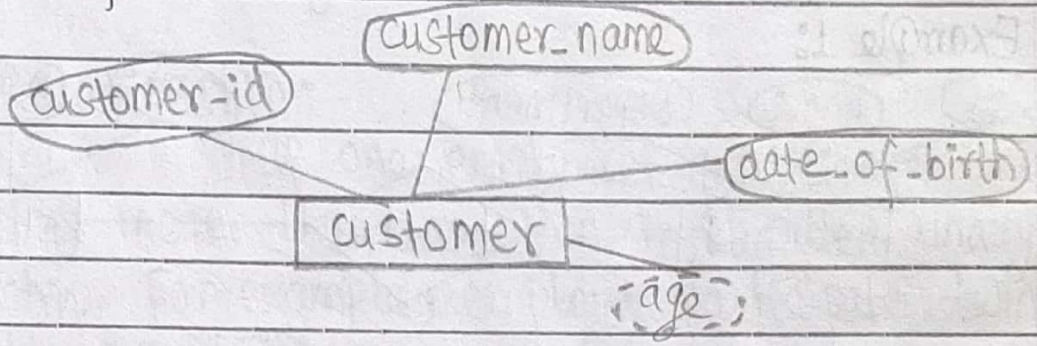
- For example, in a customer entity set attribute customer_contact is multivalued attribute, since a customer can have zero or several phone numbers.
- Multivalued attribute is represented by double ellipse.




c) Stored and Derived attributes:-

The attribute value stored in the database is called stored attribute. For example; the data_of_birth attribute of customer is stored attribute.

An attribute whose value needs not to be stored rather than it can be computed from another attributes is called derived attribute. For example; in customer entity set, attribute age is derived attribute if customer entity set has attribute date_of_birth and current date. Derived attributes are depicted by dashed ellipse.




## 3) Relationship and Relationship sets:-

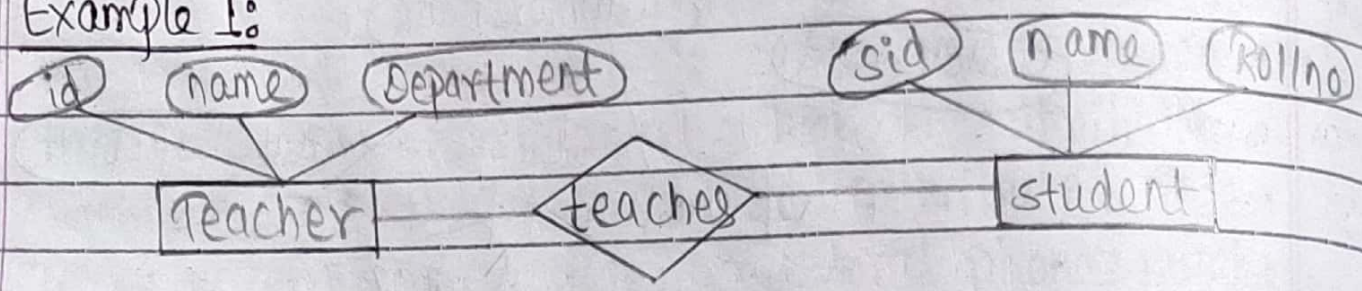
Association between two or more entities is called relationship. For example; "Arjun teaches Elina", here teaches is the association between entities Arjun and Elina.

A relationship set is a set of relationships of the same type. Formally, it is a mathematical relation on $n \geq 2$ entity sets. If $E_1, E_2, \ldots, E_n$ are entity sets, then a relationship set R is a subset of $\{(e_1, e_2, e_3, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$ where $(e_1, e_2, \ldots, e_n)$ is a relationship.

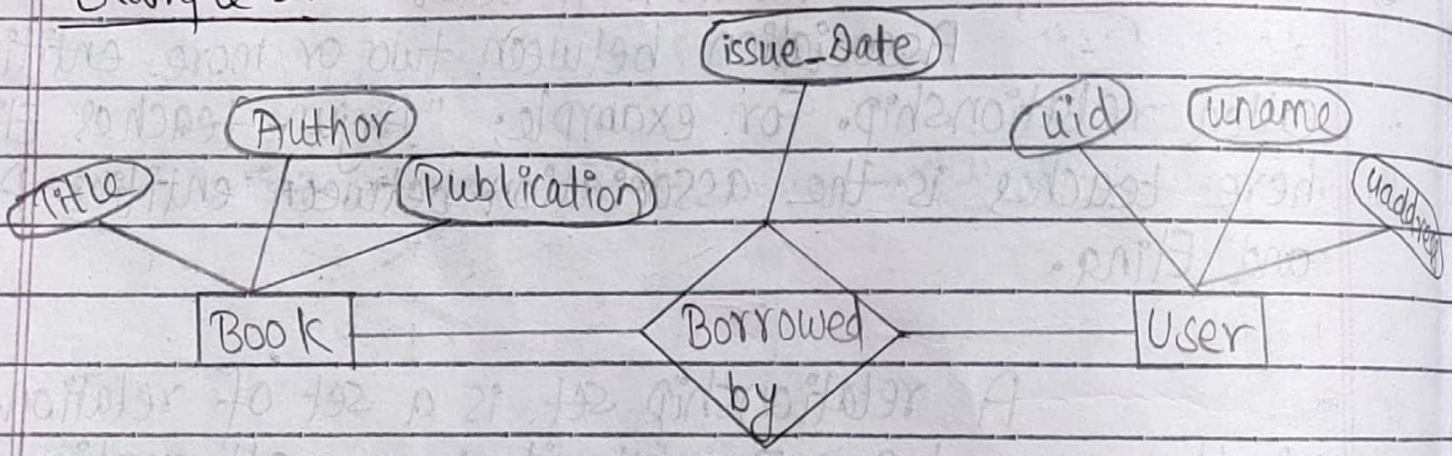
For example "Teacher teaches student", here. teaches is the association between entity sets, teacher and student.

Relationships are represented by diamond symbol in ER diagram:

Example 1:




Example 2:



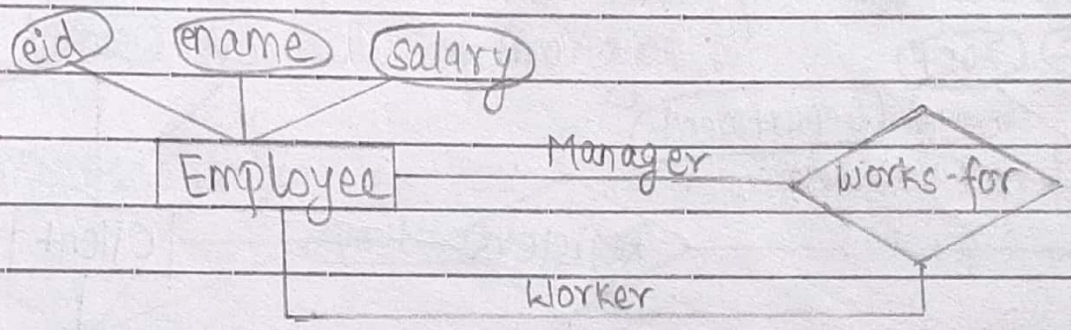
## Degree of Relationship:-

Number of entity set that participate in a relationship set is called degree of the relationship set. On the basis of degree, relationship can be divided as below:

i) Unary Relationship
ii) Binary Relationship
iii) N-ary Relationship.

### i) Unary Relationship:-

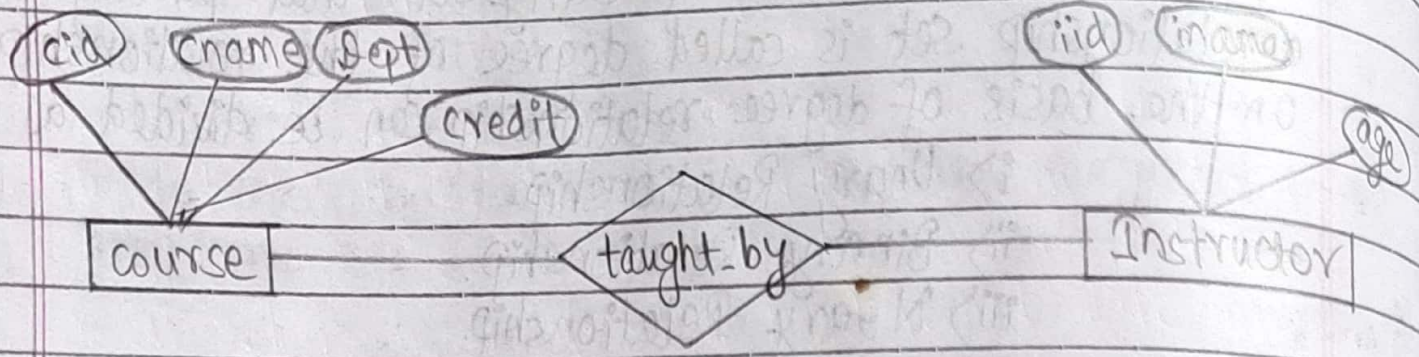
If one entity set participates in a relationship more than one then it is called unary relationship. For example, employee participate twice in relationship set works, once as a manager and again as worker.




### ii) Binary Relationship:-

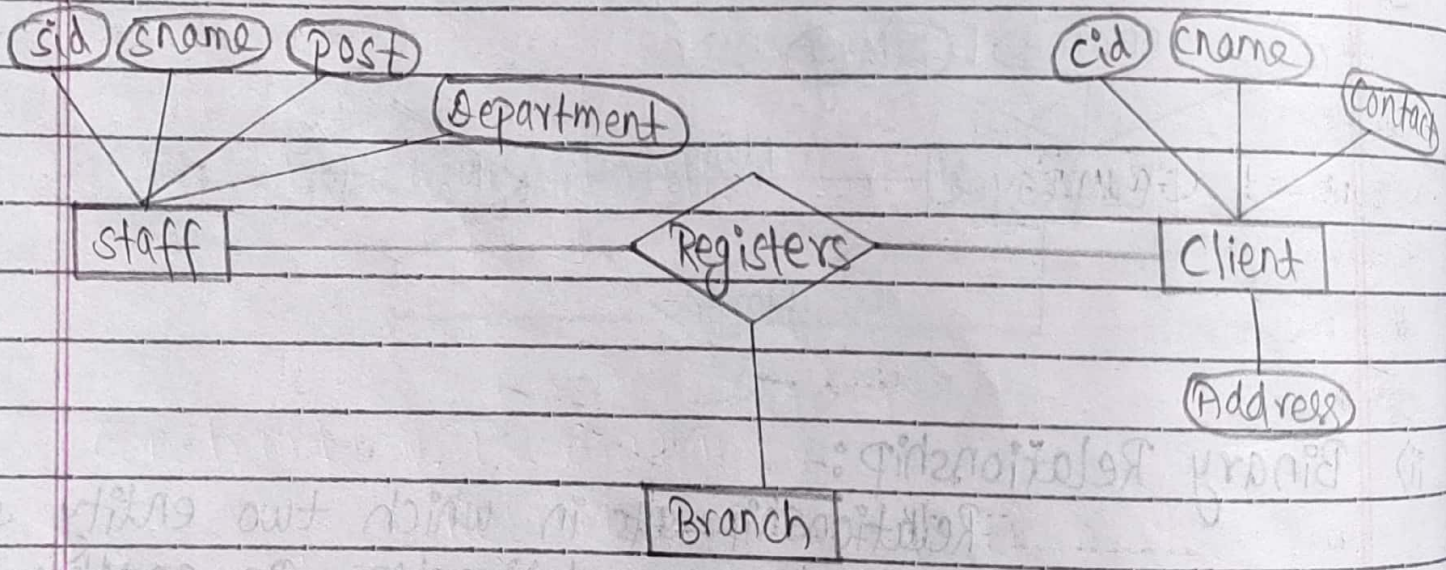
Relationship sets in which two entity sets participates are called binary relationship. In another way, the relationship of degree two are called binary relationship. This is the most common type of relationship in database system. For example; taught_by relationship given below has degree 2 and hence

it is binary relationship.




iii) N-ary Relationship:-

Relationship set which has more than two entity sets involved is called n-ary relationship. Ternary and Quaternary relationships are special cases of n-ary relationship.



## Role indicator in E-R Diagram :-

The function that an entity plays in a relationship is called that entity's role. Since entity sets participating in a relationship sets are usually distinct, thus roles are implicit and are not usually specified. In some cases, the entity set may participate in a relationship set more that once. This types of relationship set sometimes called recursive relationship set. In this type of relationship set, explicit role names are necessary to specify how an entity participates in a relationship instance.

For example; consider an entity set employee and a relationship set works_for. The first employee of the pair takes the role of worker and second takes the role of manager.

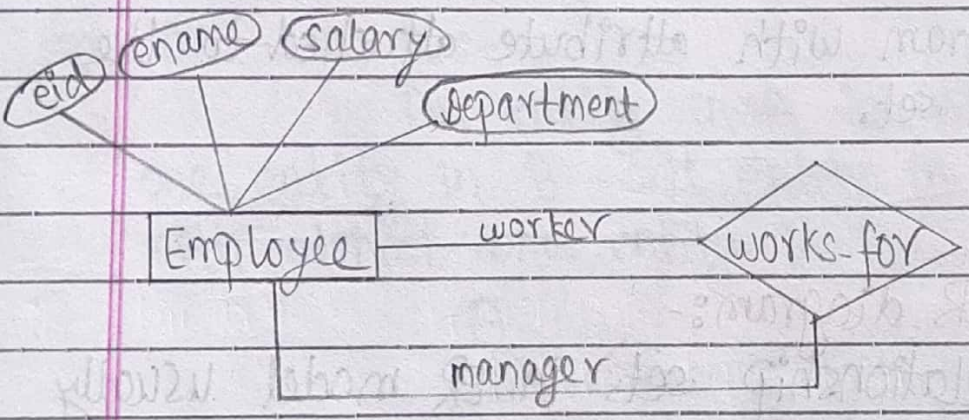



Fig:- E-R diagram with role indicator.

## Descriptive attributes :-

An attribute of the relationship set is called descriptive attributes.

For example; consider a relationship set 'deposites' with entity set customer and account. We could associate the attribute access_date to that relationship to specify the most recent date on which a customer accessed an account.

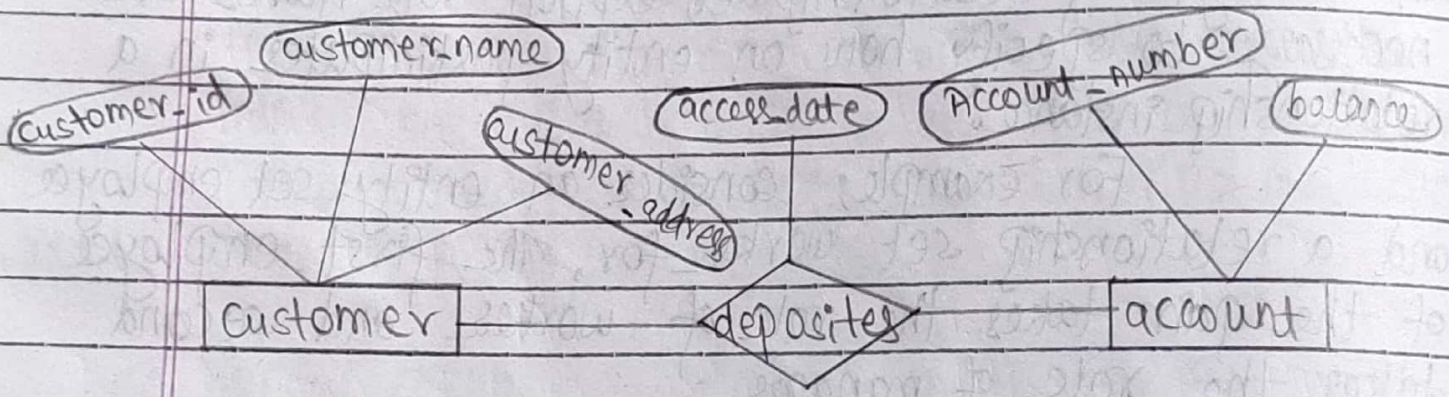



Fig :- E-R diagram with attribute attached to the relationship set.

## Constraints in ER diagram :-

Relationship sets in ER model usually have certain constraints that limit the possible combinations of entities that may involved in the corresponding relationship sets. Database contents must satisfy these constraints. The most important constraints are:

i) Mapping cardinality constraints &
ii) Participation constraints

## i) Mapping cardinality constraints :-

Mapping cardinalities expresses the number of entities to which another entity can be associated through a relationship set. It is also termed as cardinality ratio.

Mapping cardinalities are useful to describe the binary relationship set, but it can also be used to describe relationship having more than 2 entity sets. For a binary relationship set 'R' between entity sets A and B, the mapping cardinality must be one of the following:

a) One-to-one
b) One-to-many
c) Many-to-one
d) Many-to-many

## a) One-to-one :-

An entity in A is associated with at most one entity in B and entity in B is also associated with atmost one entity in A.

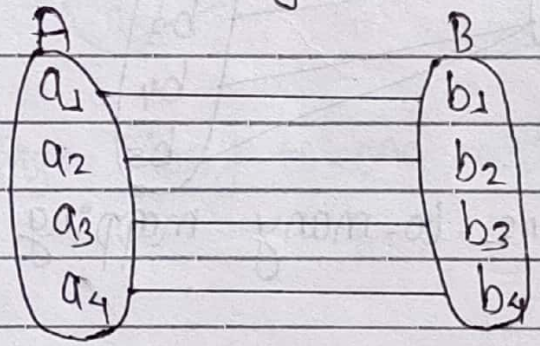
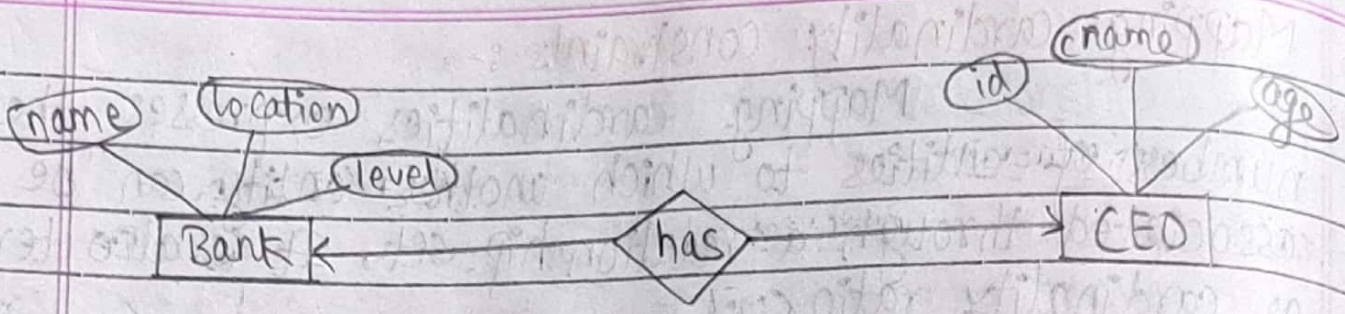



Fig :- One-to-one mapping cardinality




In above ER diagram, every bank has only one CEO and a person can be CEO of only one bank, therefore it shows one-to-one relationship between bank and CEO.

b) One-to-many:-

An entity in A is associated with any number of (zero or more) entities in B. An entity in B however can be associated with at most one entity in A.

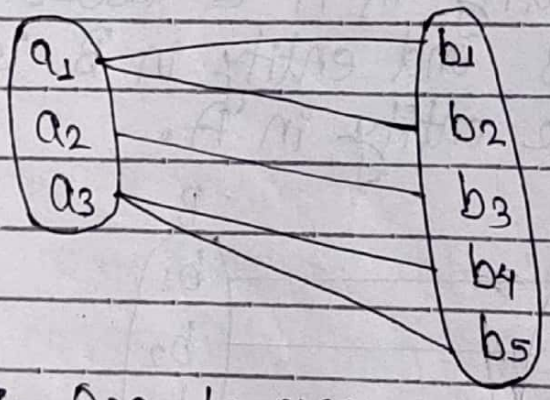
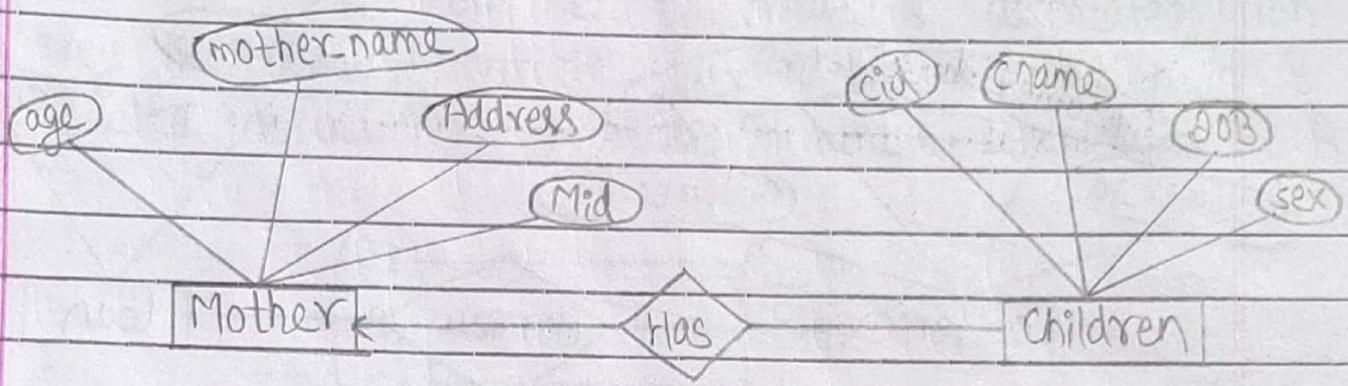



Fig:- One-to-many mapping cardinality

Example:-




In this example, a mother can have any number of children but children can have any one mother.

c) Many-to-one:-

An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.

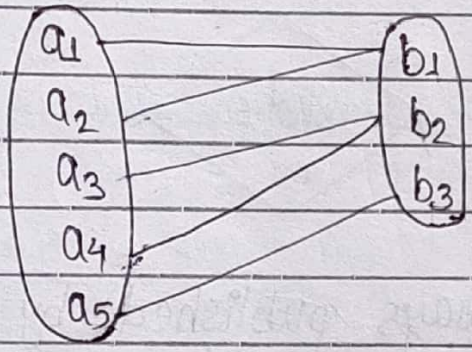
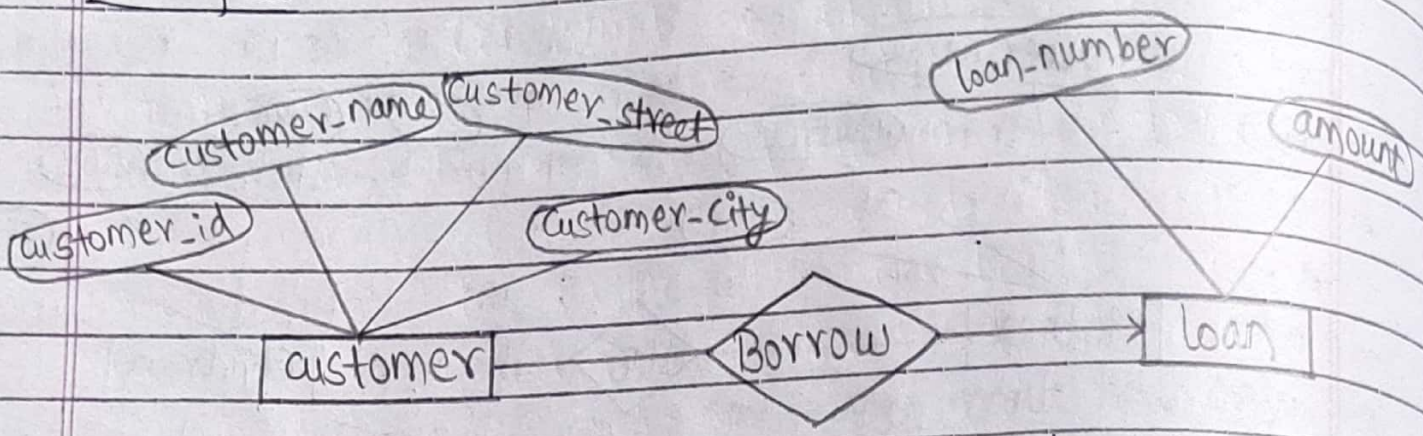


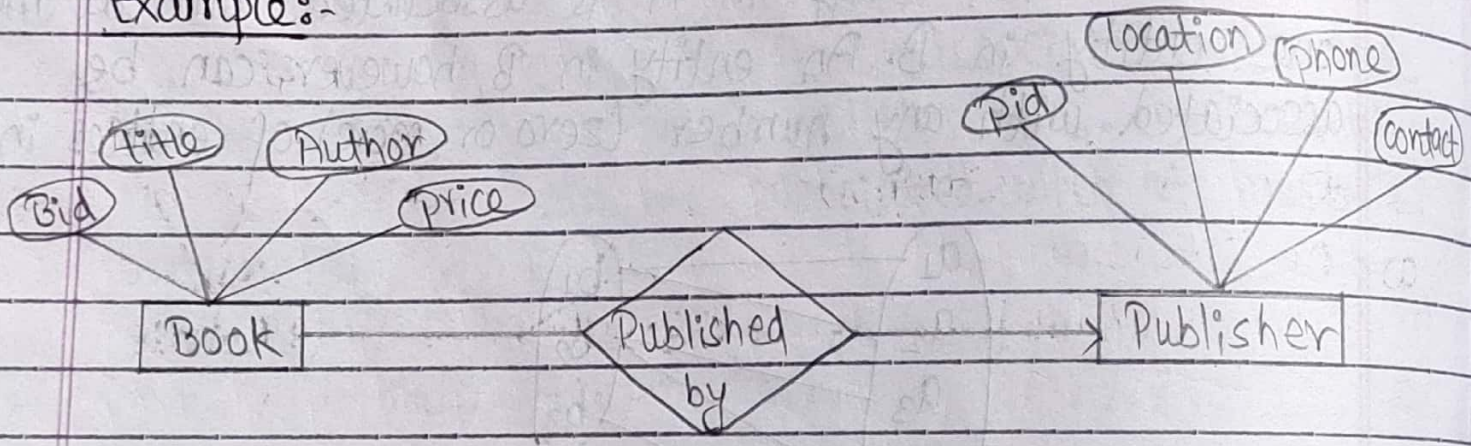

Fig:- Many-to-one mapping

Example:-




This indicates that a loan is associated with several customers via borrow but a customer is associated with at most one loan via borrow.

Example:-




A book is always published by only one publisher but a publisher can publish any number of books.

d) Many-to-Many Relationship:-

An entity in A is associated with any number of entities in B, and an entity in B is also associated with any number of entities in A.

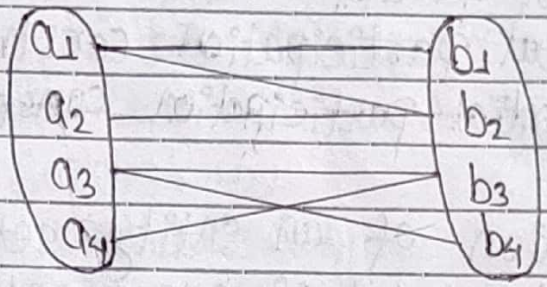
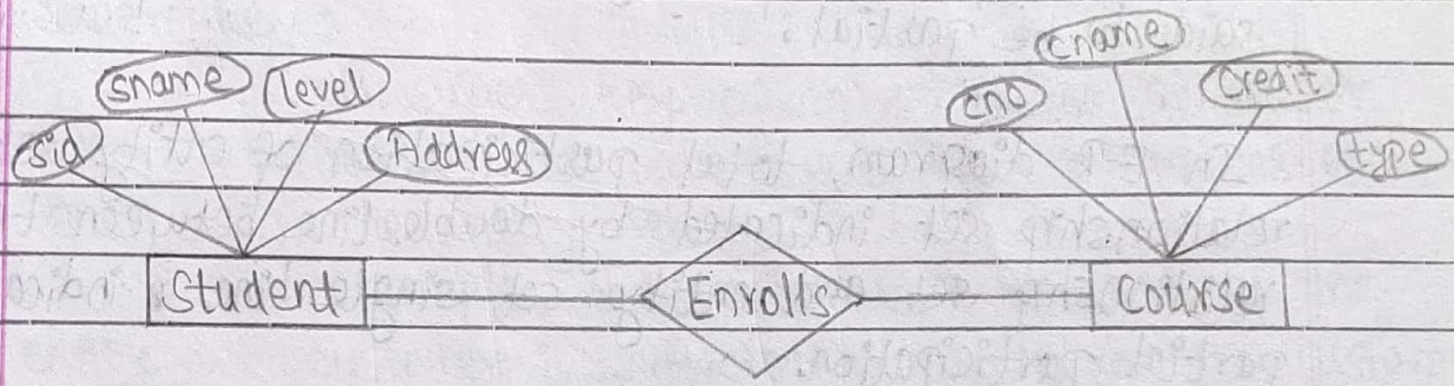



Fig:- Many-to-many mapping cardinality

Example:-




A student can enrolls in more than one subject and a subject can be enrolled by many students.

ii) Participation constraints :-

Constraints on ER model that determines whether all or only some entity occurrences participate in a relationship is called participation constraint. There are two types of participation constraints:

a) Total participation constraints
b) Partial participation constraints.

The participation of an entity set E in a relationship set R is said to total if every entity in E participates in at least one relationship in R. If only some entities in E participate in relationship set in R, the participation of entity set E in relationship R is said to be partial.

In E-R diagram, total participation of entity set in relationship set indicated by double line between that relationship set and entity set, single line indicates partial participation.

Example:-

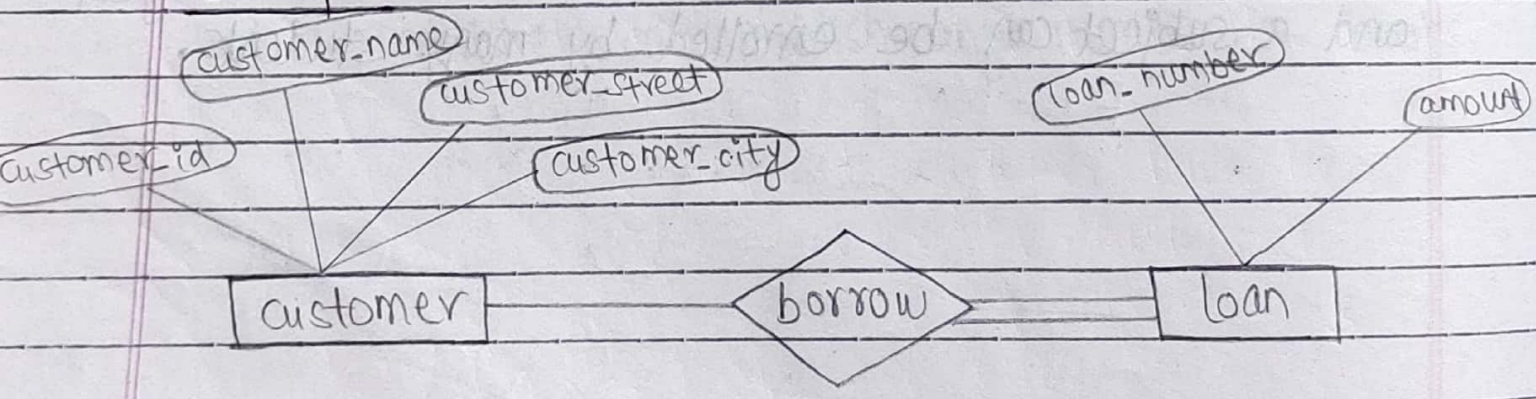



Fig:- Participation constraints in ER diagram

attribute which can find row uniquely

Date
Page

## Keys :-

Set of one or more attributes whose values are distinct for each individual entity in the entity set is called key and its values can be used to identify each entity uniquely. Keys also help uniquely identify relationship and thus distinguish relationships from each other. There are different type of keys which are:

i) Super keys
ii) Candidate key (Minimal super key)
iii) Primary key
iv) Composite key
v) Foreign key

### i) Super key :-

A super key of an entity set of one or more attributes whose values uniquely determines each entity in the entity set.

For example, the customer_id attribute of the entity set customer is sufficient to distinguish one customer entity from another. Thus, customer_id is a super key. Similarly, the combination of customer_name and customer_id is a super key for the entity set customer. The customer_name attribute of customer is not a super key, because several people might have the same name.

if we cannot break super key, then it is candidate key.

ii) Candidate key :-

A candidate key of an entity set is a minimal super key. That is a super key which does not have any proper subset is called candidate key.

For example; suppose that a combination of customer_name and customer_address is sufficient to distinguish among member of the customer entity set. Then both {customer_id} and {customer_name, customer_address} are candidate keys. Although the attributes customer_id and customer_name together can distinguish customer entities, their combination does not form a candidate key, since the attribute customer_id alone is a candidate key.

All candidate keys are super keys but vice-versa is not true.

iii) Primary key :-

A primary key is a candidate key that is choosen by the database designer as the principal means of uniquely identifying entities within an entity set. There may exists several candidate keys; one of the candidate key is selected to be the primary keys.

For example; entity set customer have two candidate keys, {customer_id} and {customer_name, customer_address}, if database designer chooses customer_id for the purpose of uniquely identifying

entities within entity set then it becomes primary key.

iv) Composite key:-

If a primary key contains more than one attribute then it is called composite key.

For example, if database designer choose customer_id as primary key then it is not composite key but if database designer choose {customer_name, customer_address} as primary key then it is composite key.

v) Foreign key:-

A foreign key is an attribute or combination of attribute that is used to establish and enforce relationship between two relation (table). A set of attributes that enforces primary key of another table is called foreign key.

For example, if a customer creates an account in a bank then account_number (primary key of account) can be used as foreign key in customer relation.

## Primary keys for Relationship sets:-

Suppose, R is a relationship set involving entity sets $E_1, E_2, \ldots, E_n$. Let us consider primary key $E_i$ is a set of attributes that form primary key in entity sets $E_i$ $(i = 0, 1, 2, \ldots, n)$. Then set of attributes primary_key (E) ∪ primary_key ($E_2$) ∪ ..... ∪ Primary_key(En) describes individuals relationship in relationship set R.

Since relationship set may also involve its own attributes, so assume relationship set consists attributes $\{a_1, a_2, \ldots, a_n\}$ then set of attributes primary_key ($E_1$) ∪ primary_key ($E_2$) ∪ ..... ∪ primary_key (En) ∪ $\{a_1, a_2, \ldots, a_n\}$ also describes individual relationship set R.

For example, let us consider an entity set 'customer' and 'account' & relationship set 'depositor' with descriptive attribute access_date. Suppose that relationship set is many to many. Then primary key of desposites relationship set is combination of primary keys of customer and account and its own attribute access_date.

## Weak Entity Sets and their representation in ER diagram:-

Entity sets that doesnot have primary key is known as weak entity set and entity set that has a primary key is known as strong entity set.

In ER diagram, weak entity set is represented by double rectangles. The discriminator of a weak entity set is represented by underline (dashed underline) attribute with a dashed line and double outlined diamond represents identifying relationship.

For example;

Consider the entity set payment.

payment = {payment_number, payment_date, payment_amount}

Here, payment number are typically sequence of numbers, starting from 1 and generated for each loan. Thus, although each payment entry is distinct, payment for different loan may share the same payment number. Thus, this entity set does not have a primary key; it is a weak entity set.

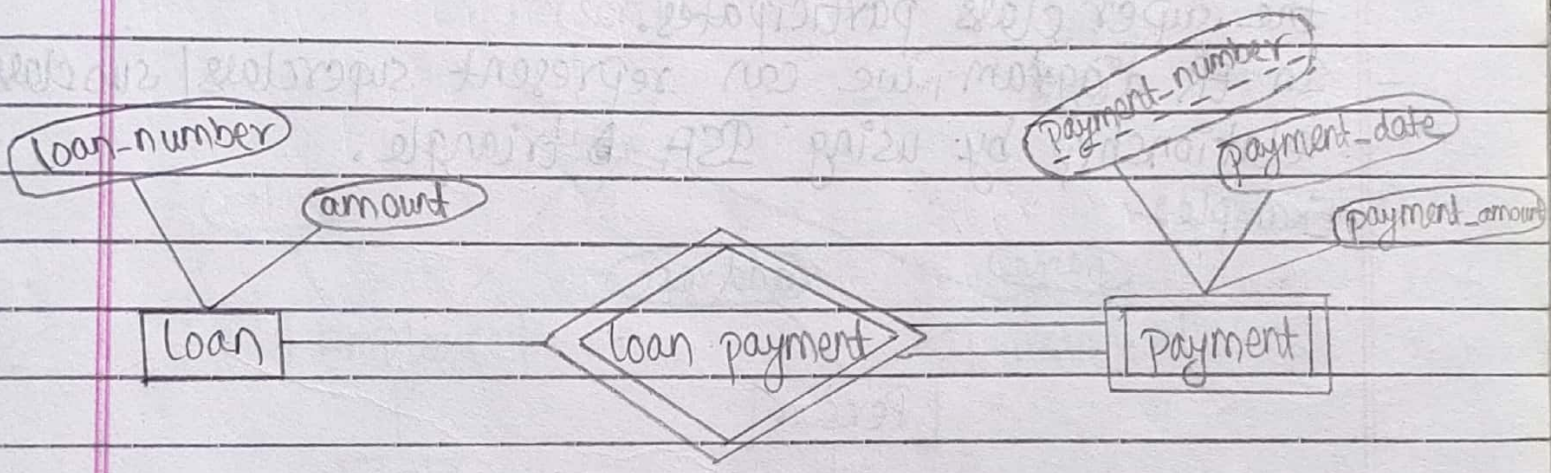



Fig:- E-R diagram with weak entity set

## Extended ER Model (EER Model):-

The EER model includes all of the concepts introduced by the ER model, with some extra features such as the concept of subclass and super-class, specialization and generalization, aggregation and categories.

### 1) Sub-class and super class:-

- A subclass is a class whose entities always be a subset of the entities in another class whereas a superclass is a grouping of subclasses.
- An entity that is a member of subclass inherits all the features from its superclass.
- The entity also inherits all the relationship in which the super class participates.
- In ER diagram, we can represent superclass/subclass relationship by using ISA triangle.

Example:-

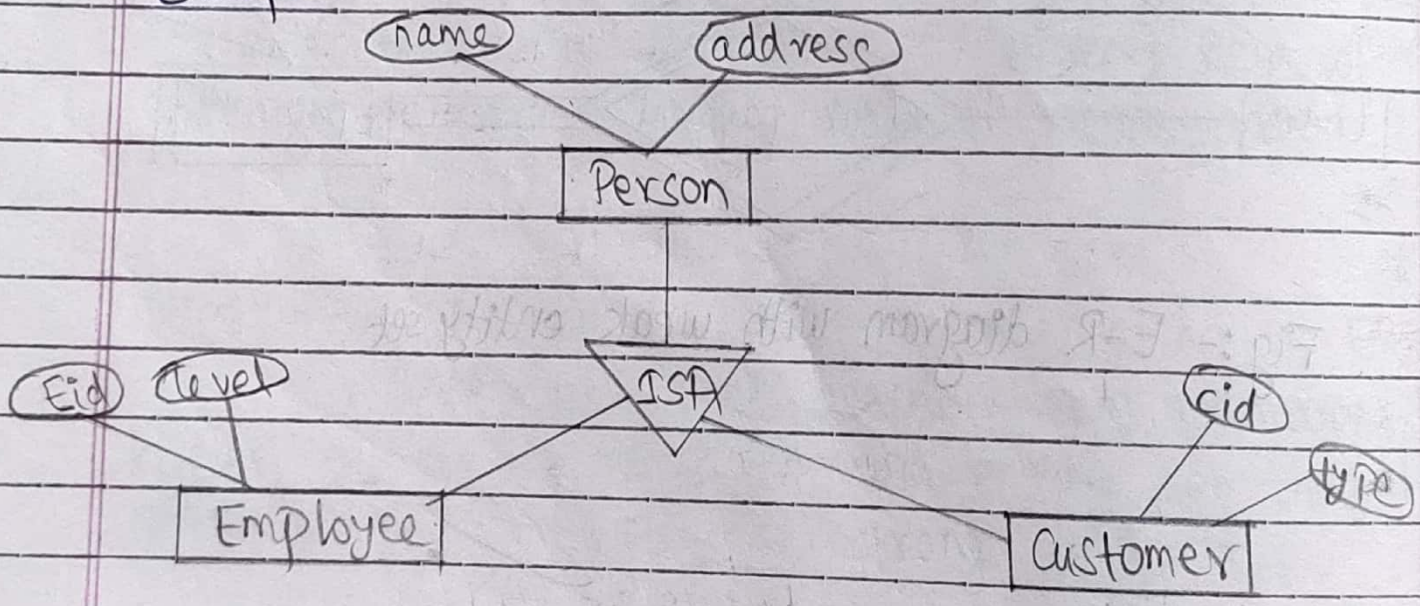



Fig:- ER diagram representing subclass and superclass

## 11) Generalization and specialization:-

- The process of defining a set of subclass from a super class is known as specialization.
- It follows top-down design approach.
- Generalization is the reverse of the specialization. It follows bottom-up approach.
- In ER diagram, generalization and specialization are represented by a triangle component labelled ISA.
- The label ISA stands for 'is a'.
- The terms specialization and generalization are used interchangeably.

Example:-

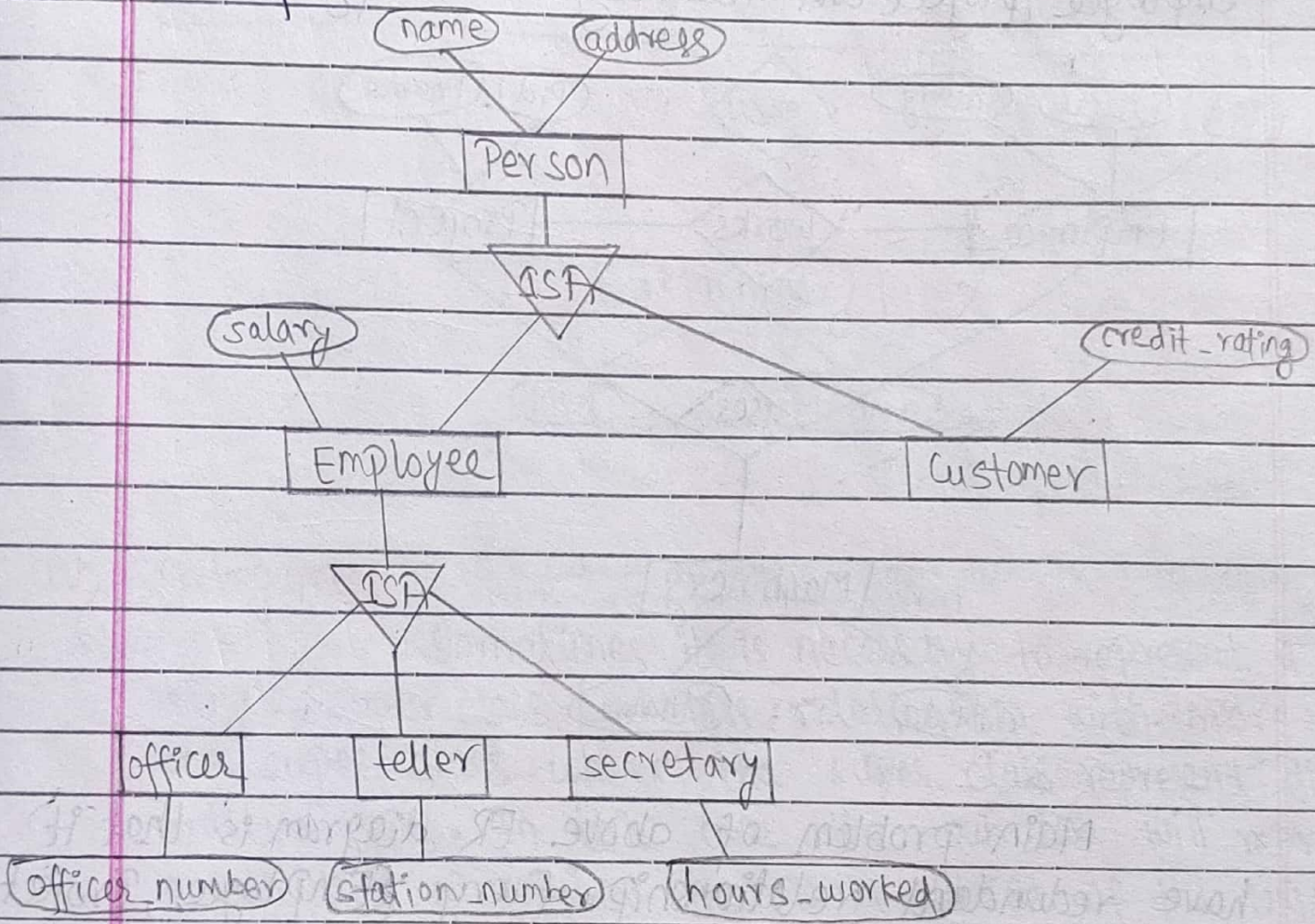


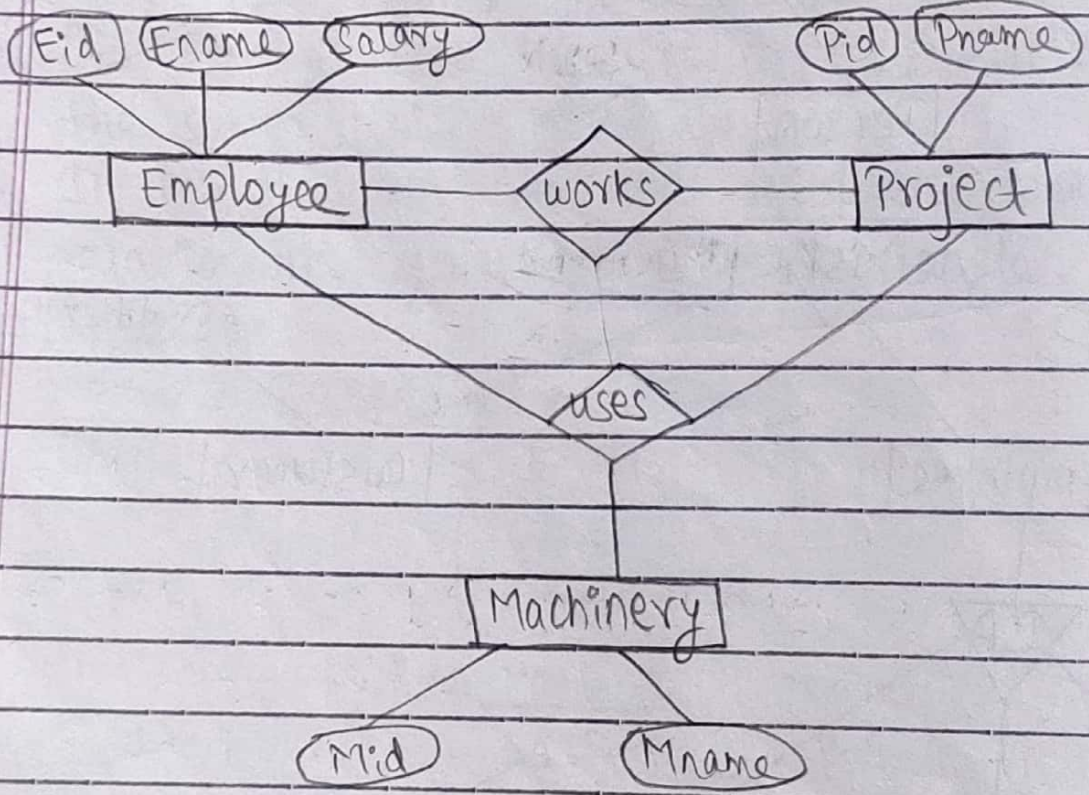

Fig:- Specialization and generalization

iii) Aggregation:-

An abstraction through which relationship sets along with associated entity sets are treated as higher level entity is called aggregation. The aggregation allow us to express relationship between relationship sets and entity sets.

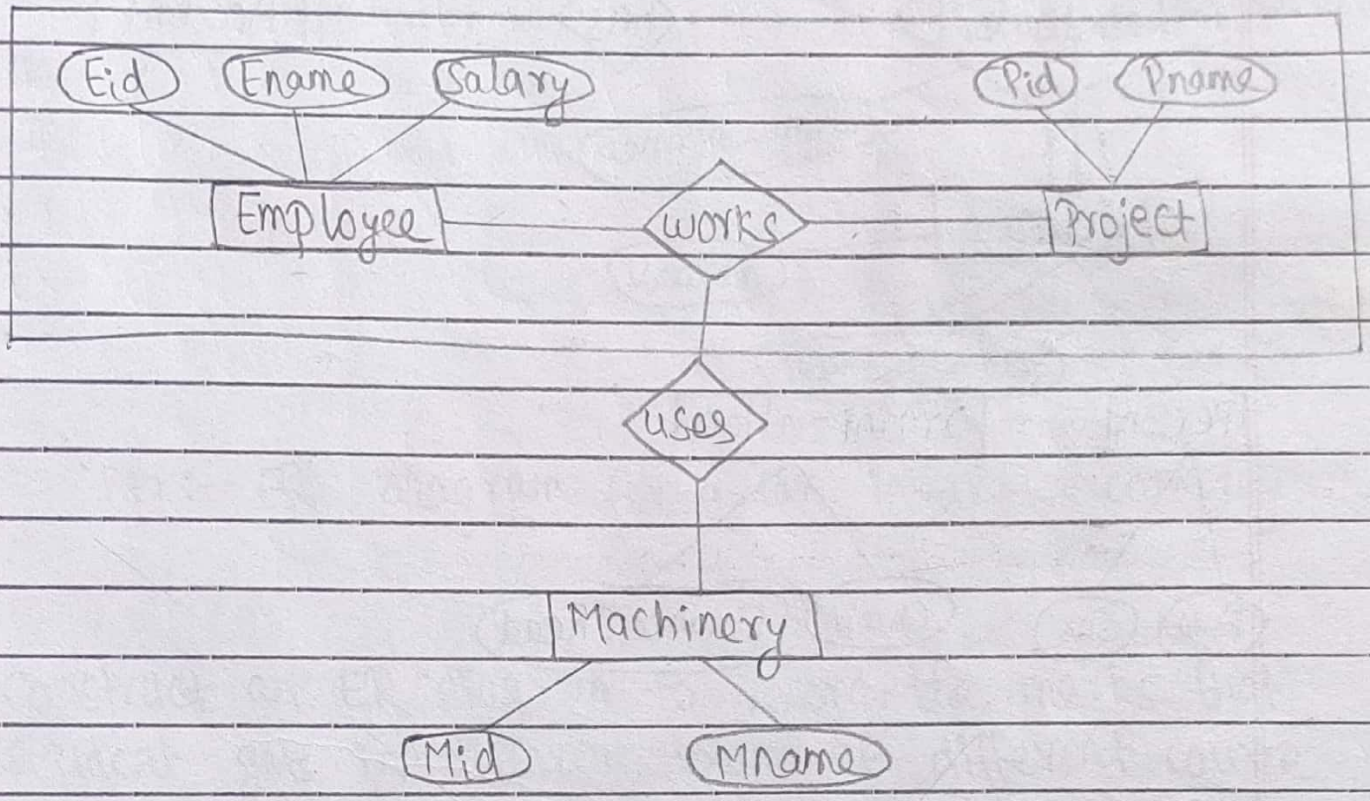
Example:-

Consider a database with information about employees who work on a particular project and use a number of machines doing that work. One alternative in this case is to define ternary relationship between employee, project and machinery as in figure below:




Main problem of above ER diagram is that it have redundant relationship. Every {Employee, Project}

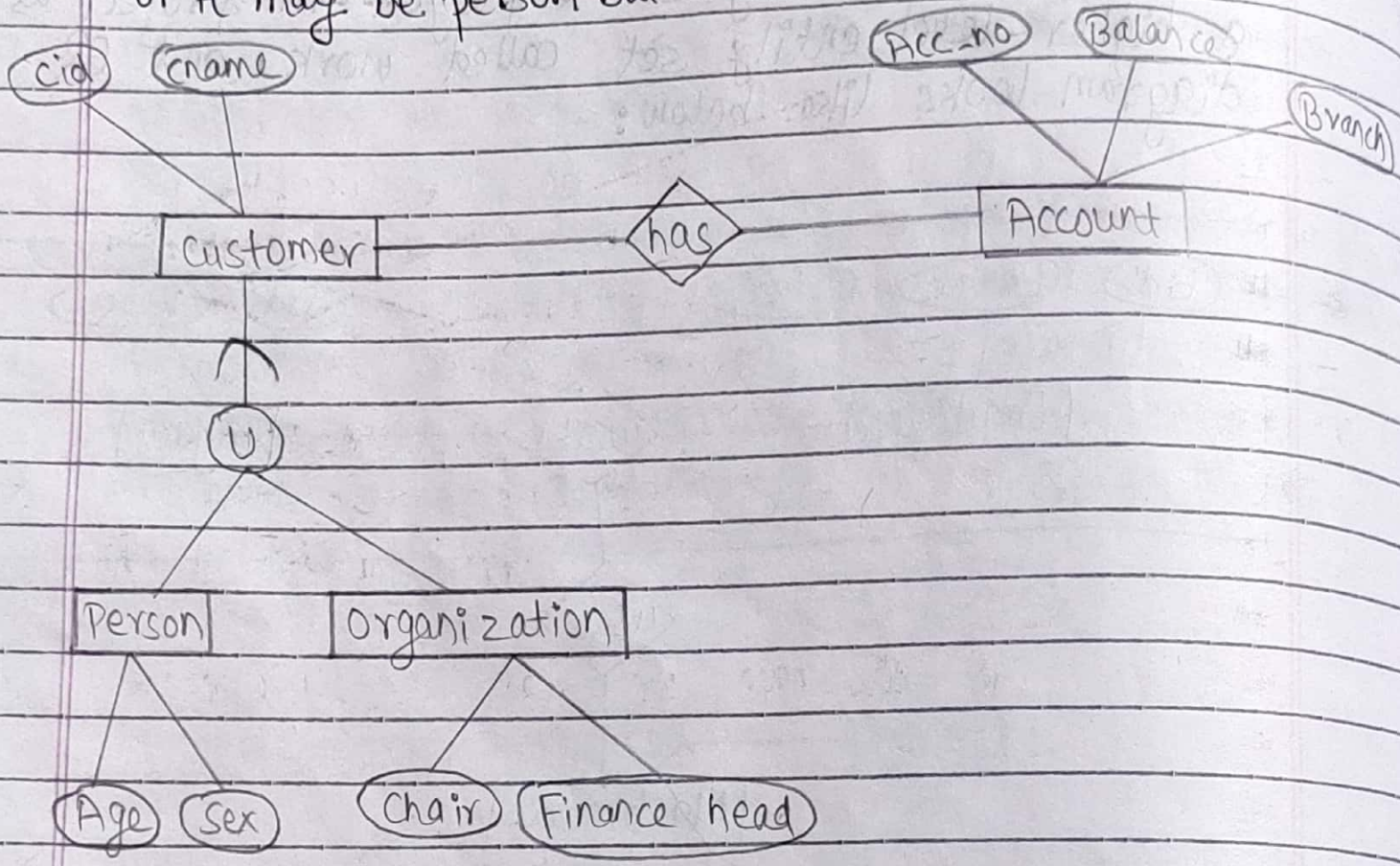
combination that appears in uses relationship set also appears in works relationship set. The solution is to use aggregation. We treat the relationship set works and the entity sets employee and project as a higher-level entity set called work and ER diagram looks like below:




IV) Categories :- (if a subclass is related to more than one super class, then that situation is represented by categories)

Sometimes it is necessary to represent a single super class/subclass relationship with more than one super class where the super class represent different entity types. In this case, the subclass will represent a collection of objects that is a subset of the UNION

of distinct entity types; we call such a subclass a union type or a category.
For example:- A customer in a bank may be organization or it may be person but cannot be both.




## Some Examples of ER diagram

1) Construct an E-R diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.

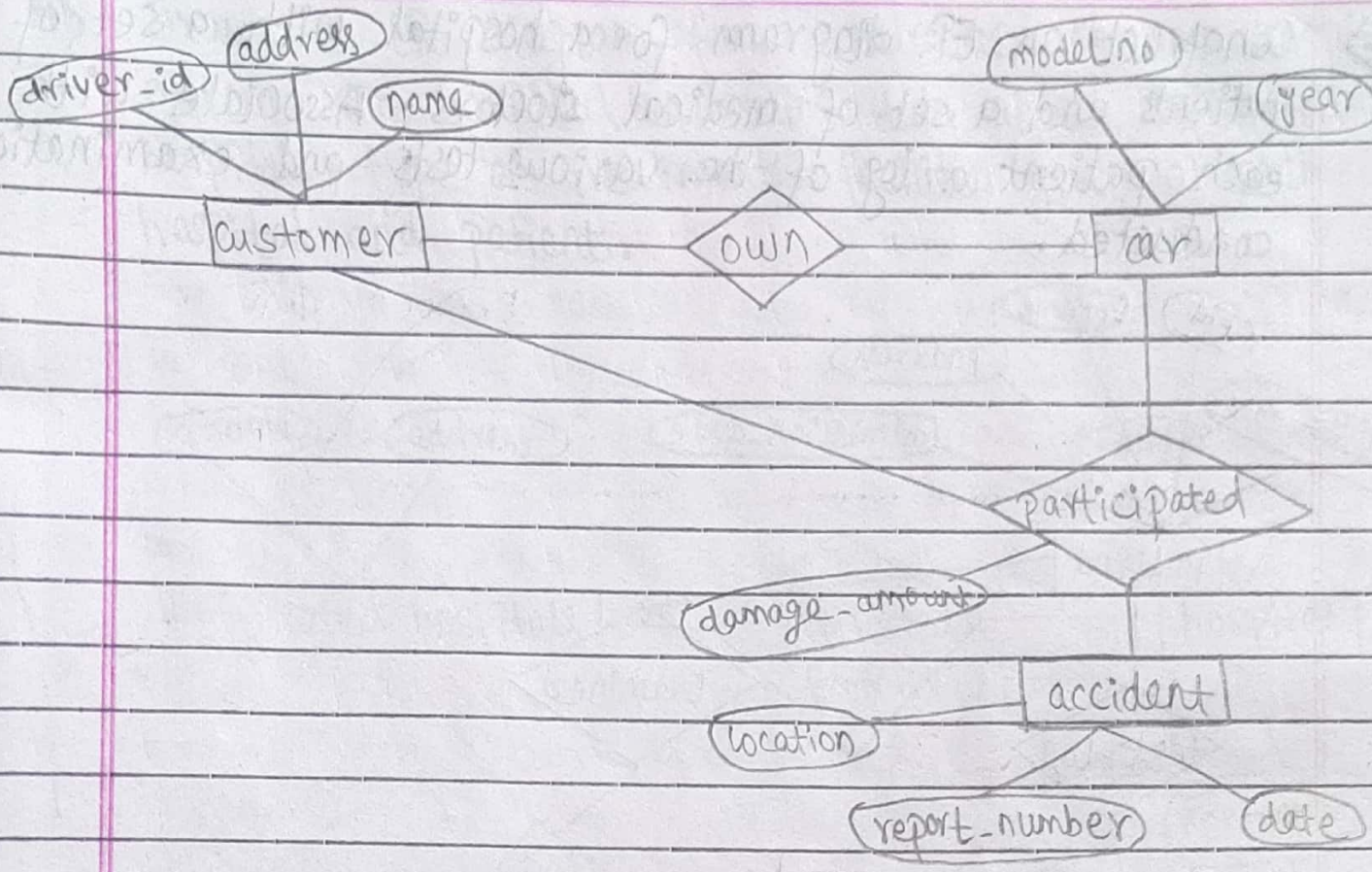



Fig :- ER diagram for a car insurance company

2) Construct an ER diagram to record the marks that student gets in different exams of different course offering.

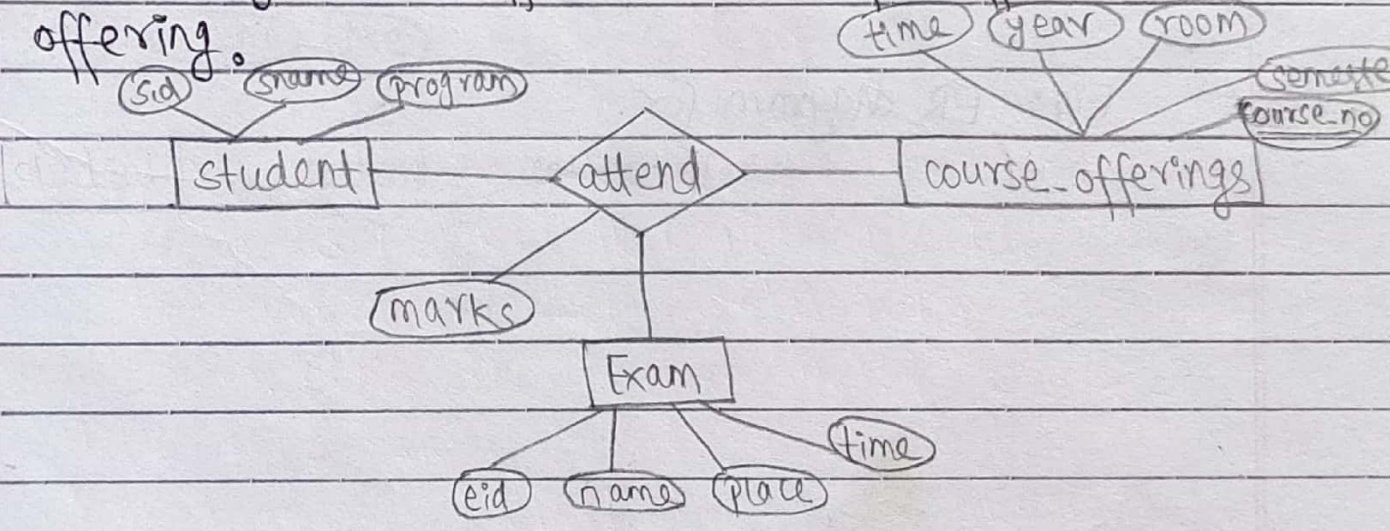



Fig :- ER diagram for marks database

3) Construct an ER diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.

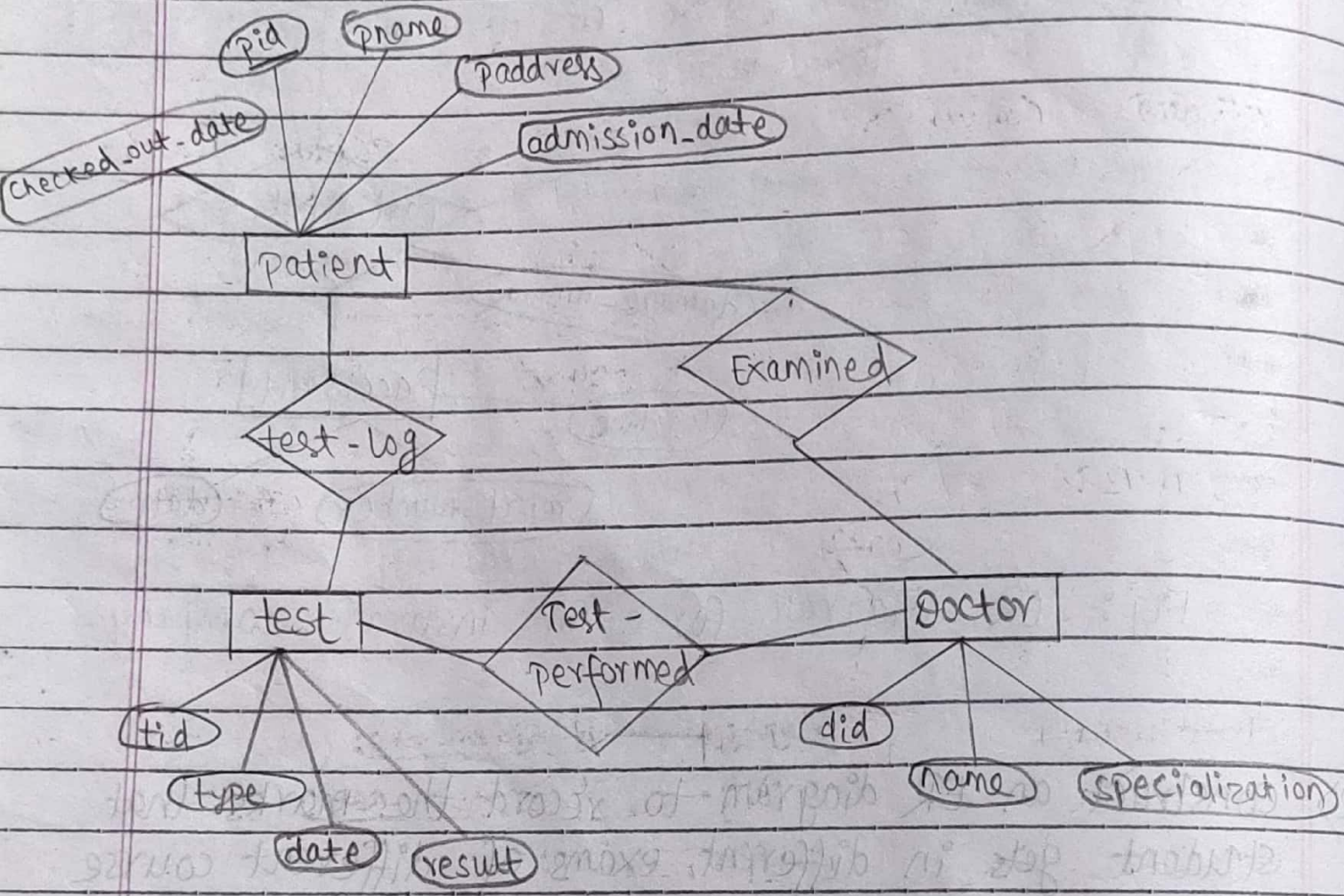
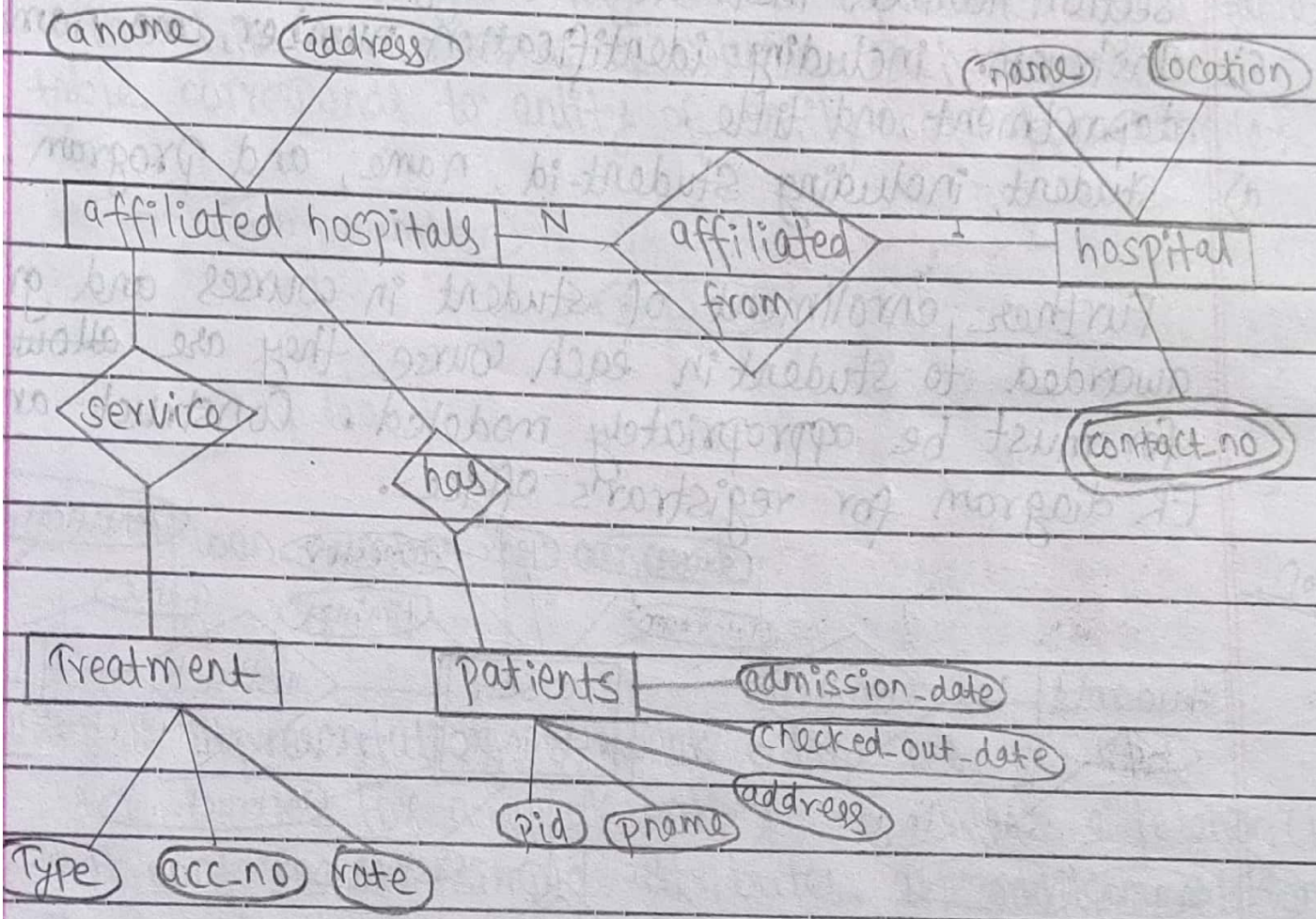



Fig:- ER diagram for

Draw an ER diagram for a database showing Hospital system. The hospital maintains data about Affiliated hospitals, type of treatments, facilities given at each hospital and patients.



5) A university registrar's office maintains data about the following entities:

a) courses, including number, title, credits, syllabus, and prerequisite.

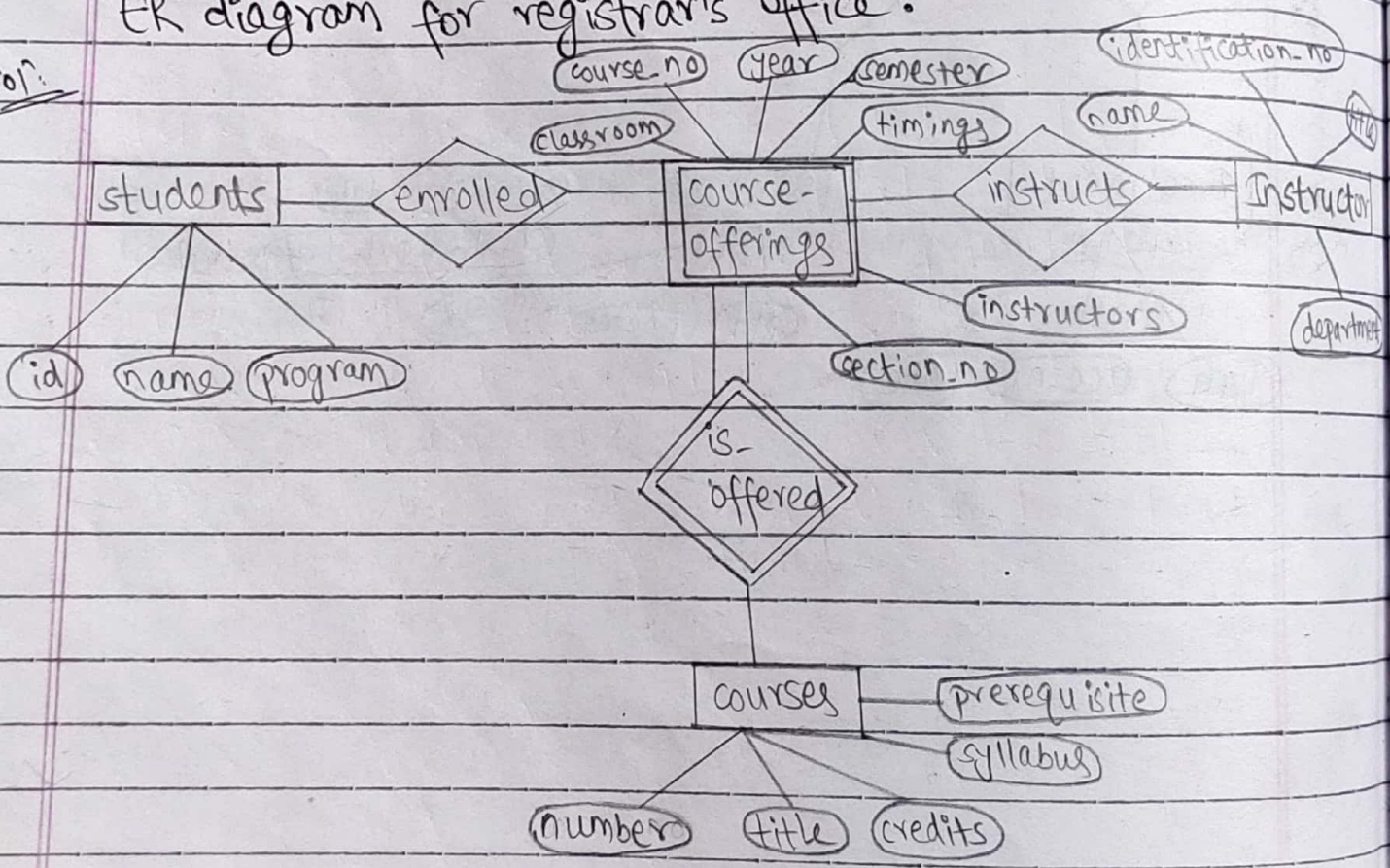
b) course offerings, including course number, year, semester, section number, instructors, timings, and class room.

c) Instructor, including identification number, name, department, and title.

d) student, including student-id, name, and program.

Further, enrollment of student in courses and grades awarded to student in each course they are enrolled for must be appropriately modeled. Construct an ER diagram for registrar's office.

Sol:

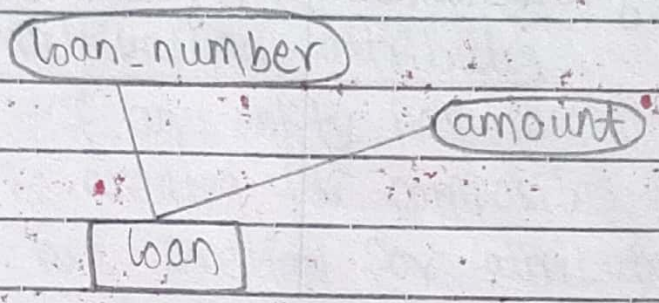


## Reduction of ER diagram to tables:

The steps to translate an ER diagram into collection of relation schema are as follows:

### Step 1: Mapping strong entity set

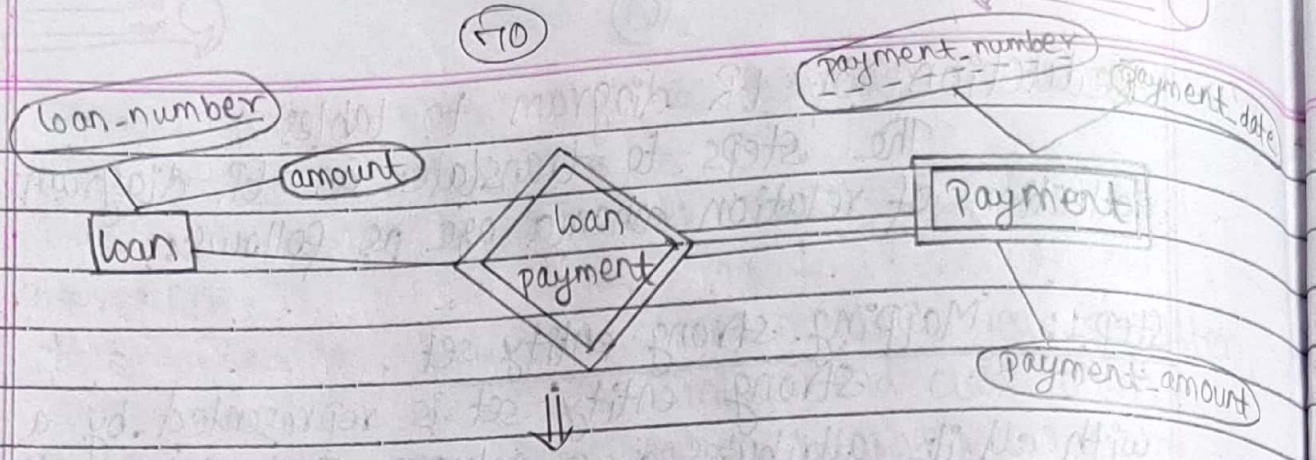
Strong entity set is represented by a table with all its attributes as a column. Each row in the table corresponds to entity of entity set. For example;




| loan_number | amount |
|---|---|
| | |

### Step 2: Representation of weak entity set

For a weak entity set, create a relation (table) that contains all simple attributes. In addition, relation for weak entity set contains primary key of the owner entity set as foreign key and its primary key is formed by combining partial key (discriminating key) and primary key of the owner entity set.

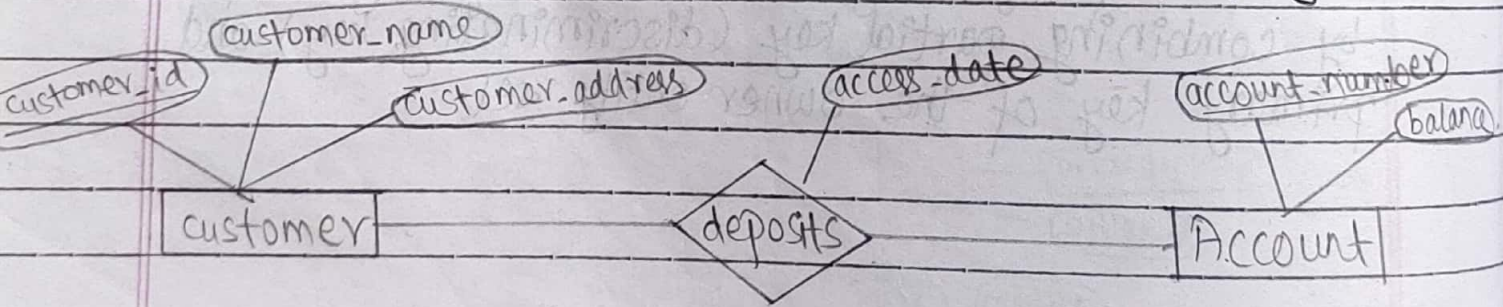



Payment

| loan_number | Payment_number | Payment_date | Payment_amount |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

## Step 3: Tabular Representation of Relationship Sets

Let R be a relationship sets with set of attributes $a_1, a_2, \ldots, a_n$ formed by union of primary key of each entity sets that participates in R. Assume that relationship set R consists of descriptive attributes $b_1, b_2, \ldots, b_n$. Now, relationship set R represented by table R with attributes $\{a_1, a_2, \ldots, a_n\} \cup \{b_1, b_2, \ldots, b_n\}$



Deposits

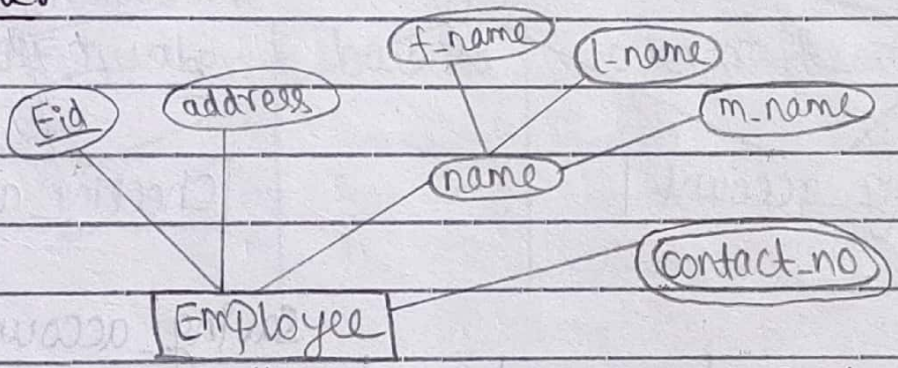
| Customer_id | account_number | access_date |
|---|---|---|
| | | |
| | | |
| | | |

Step 4: Tabular representation of composite and multivalued attributes.

If an entity has composite attributes, no separate column is created for composite attribute itself rather columns are created for attributes of composite attribute.

For a multivalued attribute, separate relation is created with primary key of the entity set and multivalued attribute itself.

Example:-




Employee

| Eid | address | f_name | l_name | m_name |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |

Contact_no

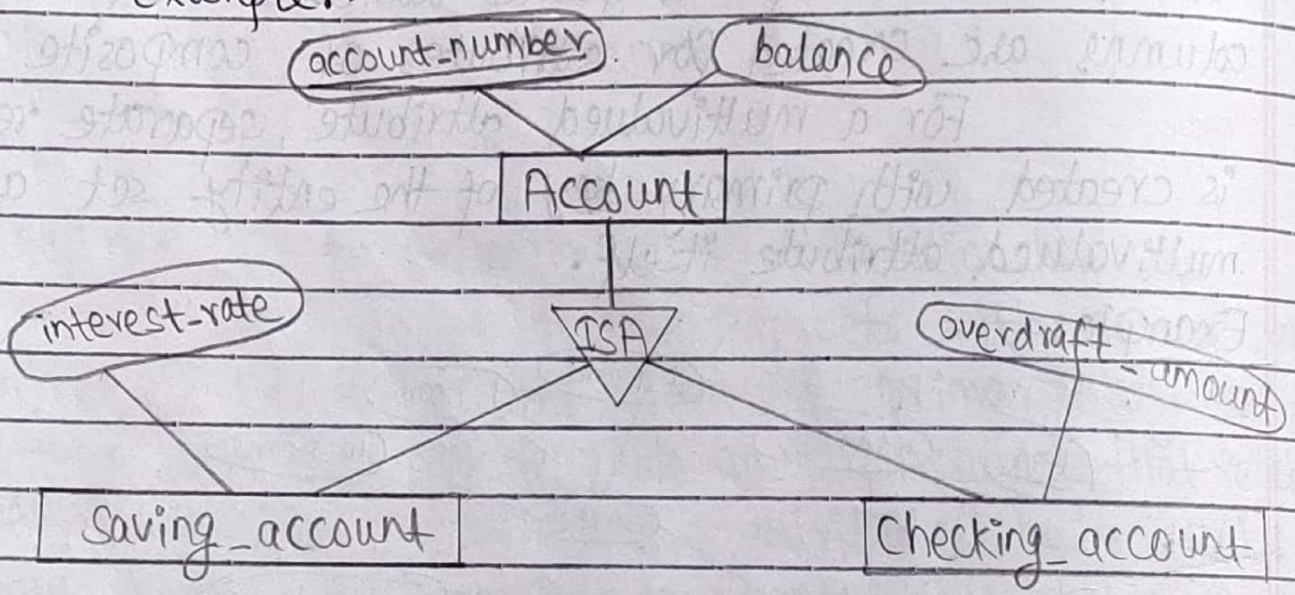
| Eid | Contact_no |
|---|---|
| | |
| | |
| | |

## Step 5: Tabular Representation of Generalization and Specialization

There are two different methods for transforming to a tabular form and ER diagram that includes generalization and specialization:

a) Create a table for higher level entity set. For each lower level entity set, create a table that includes a column for each of the attributes of that entity set plus a column for each attribute of the primary key of the higher level entity set.

Example:-




Account

| account_number | balance |
|---|---|
| | |
| | |
| | |

Saving_account

| account_number | interest_rate |
|---|---|
| | |
| | |
| | |

Checking_account

| account_number | overdraft_amount |
|---|---|
| | |
| | |
| | |

b) Another approach is to create relation only for lower level entity set.

Example:-

Saving_account

| account_number | balance | interest_rate |
|---|---|---|
| | | |
| | | |
| | | |

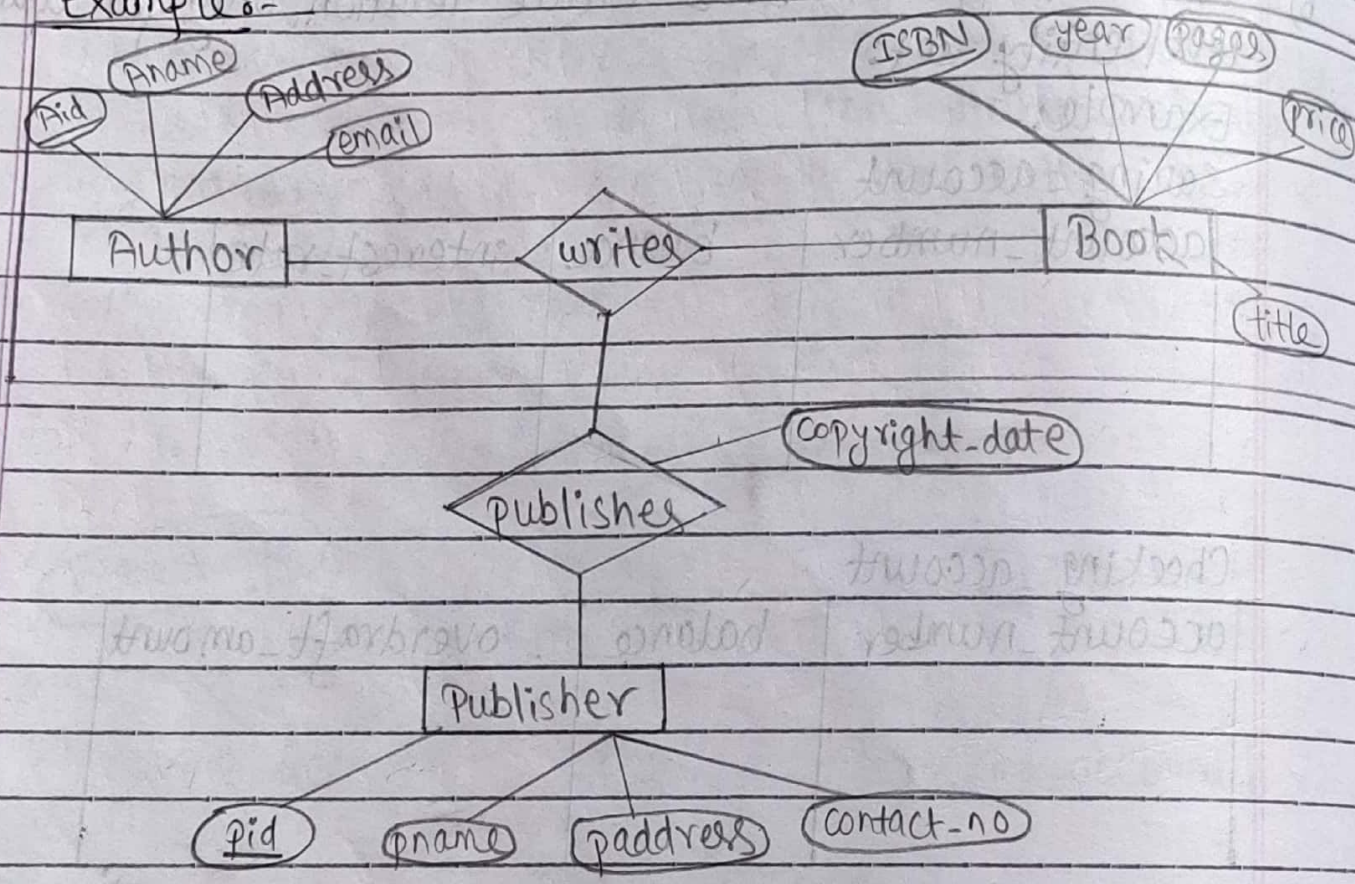
Checking_account

| account_number | balance | overdraft_amount |
|---|---|---|
| | | |
| | | |
| | | |

## Step 6 : Tabular Representation of Aggregation

In relational model, separate relation is created for this relationship set and the relation contains primary key of associated entity sets and the relationship set and its own attribute if any. Transformation of other entity sets and relationship set is similar to previously discussed.

Example :-




Publishes

| Aid | ISBN | Pid | copyright_date |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

Relation is a table.

Integrity constraint (eg):- Bank b/c should not be less than 500.
condition.

20 marks



Date
Page

# Unit 1.3: Relational Model

## Definition:

A relational database is a collection of relation or two-dimensional tables having distinct names. The relational model represents both data and relationship among those data using relation. A relation is used to represent information about any entity and its relationship with other entities in the form of attributes (or columns) and tuples (or rows)

## Relational Database Schema:-

- A relational schema consists of a relation name R and a set of attributes (or fields) $A_1, A_2, .., A_n$ is represented by $R(A_1, A_2, ..., A_n)$ and is used to describe a relation R.
- A set of relation schema as $\{R_1, R_2, ..., R_m\}$ together with a set of integrity constraints in the database constitutes relational database schema.
- For example, the relational database schema 'Book' is a set of relation schema, namely Book, Publisher, Author, written_by, and Reviewer is as shown below:

Book (<u>ISBN</u>, Title, category, Price, Pages, year, pid)
Publisher (Pid, Pname, City, email)
Author (Aid, aname, address, email)
Written_by (ISBN, Aid)

the situation in which the database currently exists - instance.



Reviewed_by ( Rid, ISBN, Rating)
Reviewer (Rid, rname, raddress, remail)

## Relational Database Instance:

- A ~~relational~~ database instance of a relational database schema $S = \{R_1, R_2, \ldots R_m\}$ is a set relation instances $\{r_1, r_2, \ldots, r_m\}$ such that each relational instance $r_i$ is a state of corresponding relational schema $R_i$.
- In addition, each relation instance must satisfy the integrity constraints specified in the relational database schema.
- For example, relational database instance corresponding to the Publisher relation of Book schema is as shown below:

Publisher

| Pid | pname | city | email |
|---|---|---|---|
| 01 | KEC | ktm | kec@gmail.com |
| | | | |
| | | | |

## Schema Diagram:-

Schema diagram is a graphical representation of database schema along with primary key and foreign key dependencies.

In schema diagram, each relation is represented by box where attributes are listed inside box and relation name is specified above it. Primary key in relation is placed above the horizontal line that crosses the box. Foreign key in schema diagram appears as arrow from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

Example:

Figure below shows the schema diagram for banking enterprise.

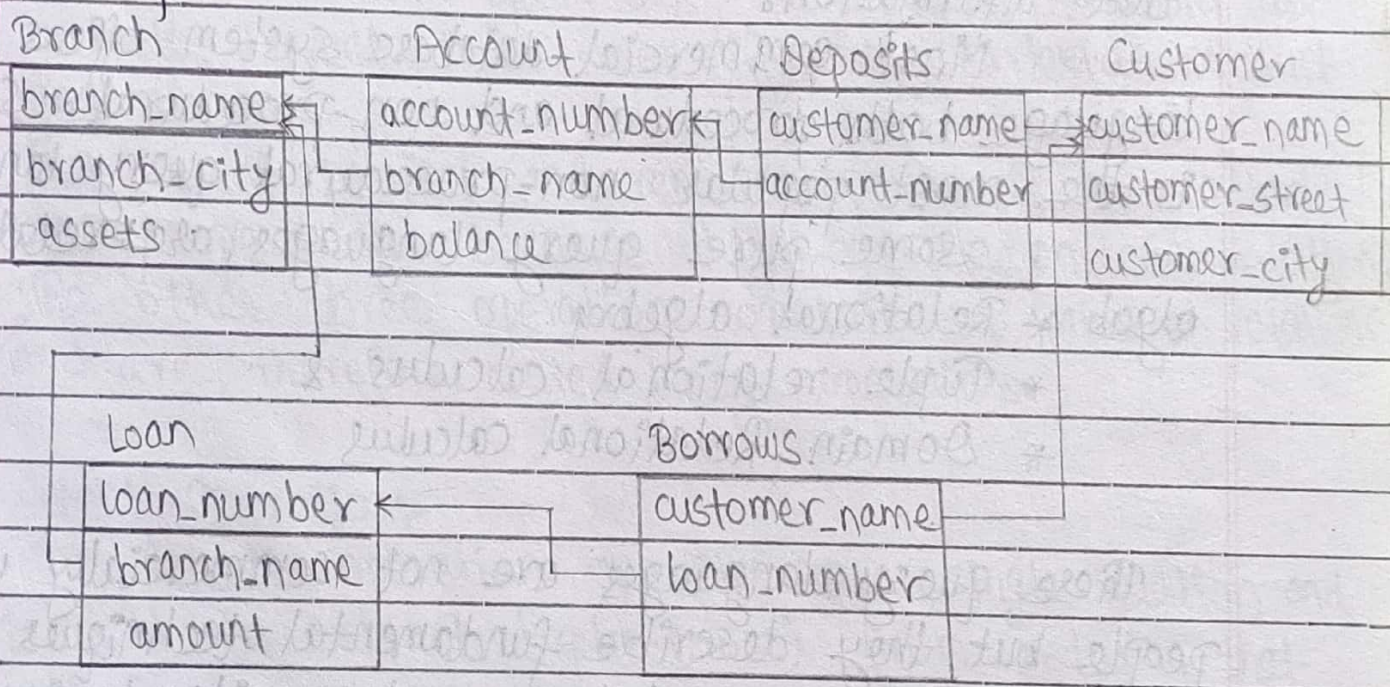



Fig:- Schema diagram for Banking enterprise

The difference between ER diagram and schema diagram is that, in ER diagram we do not show foreign key but in schema diagram we show it explicitly.

used to manage data in database

## Query Languages :-

A query language is a language through which user request information from database. Query languages can be categorized into procedural and non-procedural. In procedural query language, user required to specify sequence of operations to the systems to compute desired information whereas in non-procedural query language user need to specify required information without specifying special procedures for obtaining that information.

Most commercial database system offers query language both procedural and non-procedural. SQL is the most popular non-procedural query language.

Some pure query languages are :- relational algeb
* Relational algebra
* Tuple relational calculus &
* Domain Relational calculus

These query languages are not commercially used by people but they describe fundamental techniques for extracting data from database and provide basics for commercial languages.

## Relational Algebra

- A relational algebra is a procedural query language.
- It consists of set of operations that take one or two relations as input and produce a new relation as their result.
- The relational algebra is very important for several reasons:

(i) First it provides formal foundation for relational model operations.

(ii) It is used as a basic for implementing and optimizing queries in the query processing and optimization modules.

## Fundamental operations of Relational algebra:

- The fundamental operations in relational algebra are select, project, union, set difference, cartesian product and rename.
- The select, project and rename operations are called unary operations because they operate on one relation. The other three operations operate on pairs of relation and are, therefore, called binary operations.

### 1) Select Operation:-

- Selection operation operates on a single relation and is used to extract tuples (rows) from relation that satisfy a given predicate.
- It is denoted by lowercase Greek letter 'σ'.

- The predicate appears as subscript of sigma (σ).

Syntax:-

$\sigma_{<predicate>}(R)$

where,

R is a relation name

Predicate is a set of conditions on the basis of which rows are selected.

Example:

Select those tuples of the loan relation whose branch is ktm.

Sol^n:

Here,

Loan (loan_number, branch_name, amount)

$\sigma_{branch\_name = ktm}(Loan)$

Comparison and logical operator in selection:

- Comparison operators ($<, >, \leq, \geq, =, \neq$) can be used to specify conditions required for selecting tuples from a relation.

- Logical operators: AND ($\wedge$), OR ($\vee$) and NOT ($\neg$) can be used to combine two or more conditions.

- The order in which conditions are applied has no significance i.e. selection conditions can be applied in any order.

Example:- Find all the tuples in which loan amount is greater than 20,000 and branch is 'ktm'.

Sol:

$\sigma_{amount > 20000 \wedge branch\_name = 'ktm'}(Loan)$

ii) The Project operation :-

- The project operation retrieves tuples for specified attributes of a relation.
- It eliminates duplicate tuples in a relation.
- The projection is denoted by uppercase Greek letter Pie ($\Pi$).
- We need to specify attributes that we wish to appear in the result as a subscript to Pie.

Syntax:

$$\Pi_{a_1, a_2, \ldots, a_n}(R)$$

where,

- R is a relation name &
- $a_i$ $(i = 1, 2, 3, \ldots, n)$ is an attributes

Example:

Find the account number and their balance from account relation.

Sol:

Here, the account relation has following schema.

Account (account_number, branch_name, balance)

$\Pi_{account\_number, balance}(Account)$

## Composition of Relational algebra expression:

For the complicated query, the relational algebra expression can be combined to form complex relational algebra expression.

Example:

Find the names of all customers who live in Kathmandu.

Sol:

Here,

customer (customer_name, customer_city, customer_street)

$\Pi_{customer\_name}(\sigma_{customer\_city='ktm'}(customer))$

## (iii) Set Difference operation:-

- The set difference operation denoted by '−', allows us to find tuples that are in one relation but are not in another.
- Formally, let r and s are two relations then their difference r−s is defined as:

$$r - s = \{ t \mid t \in r \text{ and } t \notin s \}$$

- The set difference must be taken between compatible relations.
- For r−s to be valid, it must hold
  i) r and s must have the same arity (no. of column in the table) &
  ii) Attributes domain of r and s must be compatible.

small letter - instance
big letter - schema
{r = r(R)} same both



Date
Page

Example: Find the names of all customers of a bank who have account but not loan.

Soln:

Deposits (customer_name, account_number)
Borrows (customer_name, loan_number)

$\Pi_{customer\_name}$ (Deposits) − $\Pi_{customer\_name}$ (Borrows)

## iv) Union operation:-

- The union of relation r and s is denoted by r∪s and it is the set of tuples that are either in R or in S or in both.
- It is binary operation.
- Formally, we can define union as follows:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

where,

r and S are either database relation or relation result set and t is a tuple.

- To find the union of two relations, they must be union compatible.
- Relations are set to be union compatible if
  i) Arity of a relation is same i.e. no. of fields in both relation must be same.
  ii) Domain of $k^{th}$ attribute of first relation is same as the domain of $k^{th}$ attribute of $2^{nd}$ relation.

Example: Find the names of all customers who have either an account or a loan or both.

Sol:

Borrows (customer_name, loan_number)
Deposits (customer_name, account_number)

$\Pi_{customer\_name}(Deposits) \cup \Pi_{customer\_name}(Borrows)$

Unit V)

## Cartesian product operation:-

- The cartesian product operation allows us to combine information from any two relation.
- It is denoted by cross (X).
- The cartesian product of two relations r and s denoted by r x s returns a relation instance whose schema contains all the fields of r and followed all the fields of s.
- The result of r x s combines the tuples <r, s> (combination of tuples of r and s) for each pair tuples $t \in r$, $q \in s$.
- Formally,

$$r \times s = \{ \langle t, q \rangle \mid t \in r \text{ and } q \in s \}$$

Example: Find the names of all customer who have loan at Kathmandu branch.

Sol:

Borrows (customer_name, loan_number)
Loan (loan_number, branch_name, amount)

$$\Pi_{customer\_name}\left(\sigma_{loan.loan\_number = Borrow.loan\_number}\left(\sigma_{loan.branch\_name = 'ktm'}(Borrows \times Loan)\right)\right)$$

Process:-

Let Borrows and Loan ~~reali~~ relation have following instances:

Borrows

| customer_name | loan_number |
|---|---|
| X | L01 |
| Y | L02 |

Loan

| loan_number | branch_name | amount |
|---|---|---|
| L01 | ktm | 7000 |
| L02 | Pkr | 9000 |

i) Borrows X Loan

| customer_name | Borrows loan_number | Loan loan_number | branch_name | amount |
|---|---|---|---|---|
| X | L01 | L01 | ktm | 7000 |
| X | L01 | L02 | pkr | 9000 |
| Y | L02 | L01 | ktm | 7000 |
| Y | L02 | L02 | Pkr | 9000 |

ii) $\sigma_{branch\_name = 'ktm'}(Borrows \times Loan)$

| customer_name | Borrows loan_number | Loan loan_number | branch_name | amount |
|---|---|---|---|---|
| X | L01 | L01 | ktm | 7000 |
| Y | L02 | L01 | ktm | 7000 |

iii) $\sigma_{\text{Borrows.loan\_number} = \text{loan.loan\_number}}\left(\sigma_{\text{branch\_name} = \text{'ktm'}}\left(\text{Borrows} \times \text{Loan}\right)\right)$

| customer_name | Borrows. loan_number | Loan. loan_number | branch_name | amount |
|---|---|---|---|---|
| X | L01 | L01 | ktm | 7000 |

iv) $\Pi_{\text{customer\_name}}\left(\sigma_{\text{Borrows.loan\_number} = \text{loan.loan\_numbe}}\left(\sigma_{\text{branch\_name} = \text{'ktm'}}\left(\text{Borrows} \times \text{Loan}\right)\right)\right)$

| customer_name |
|---|
| X |

## VI) Rename operation:-

- The results of relational algebra expressions donot have a name that we can refer to them.
- Rename operation is used to provide names to such relations so that we are able to refer to them at later when needed.
- It is unary operator and denoted by symbol rho ($\rho$).
- Rename operation can be used to rename ~~operations~~ relations as well as attributes in the following ways:

i) $\rho_x(E)$ returns the result of relational algebra expression E in the table name x.

ii) $\rho_x(A_1, A_2, A_3, \ldots A_n)(E)$

- Returns the result of relational algebra expression E in the table name x with the attribute renamed to $A_1, A_2, A_3, \ldots, A_n$.

- Rename operation is mainly useful in the situation where we need to find the cartesian product of a relation with itself.

Example:-

Find the largest account balance in a bank.

⇒ $\Pi_{\text{Account.balance}} - \Pi_{\text{Account.balance}}\left(\sigma_{\text{Account.balance} < \text{d.balance}}\left(\text{Account} \times \rho_d(\text{Account})\right)\right)$

(यहाँ account table लाई copy गरी 'd' rename गरेको हो।)

- Process:

Let us consider the account relation has following instance.

Account

| Account_no | balance |
|---|---|
| A01 | 7000 |
| A02 | 8000 |
| A03 | 6000 |

$\rho_d$ (Account)

| Account_no | balance |
|---|---|
| A01 | 7000 |
| A02 | 8000 |
| A03 | 6000 |

i) Account × ($\rho_d$ (Account))

| Account.Acc_no | Account.balance | d.account_number | d.balance |
|---|---|---|---|
| A01 | 7000 | A01 | 7000 |
| A01 | 7000 | A02 | 8000 |
| A01 | 7000 | A03 | 6000 |
| A02 | 8000 | A01 | 7000 |
| A02 | 8000 | A02 | 8000 |
| A02 | 8000 | A03 | 6000 |
| A03 | 6000 | A01 | 7000 |
| A03 | 6000 | A02 | 8000 |
| A03 | 6000 | A03 | 6000 |

(ii) $\sigma_{Account.balance < d.balance}$ (Account × $\rho_d$ (Account))

| A | | | |
|---|---|---|---|
| A01 | 7000 | A01 | 8000 |
| A03 | 6000 | A01 | 7000 |
| A03 | 6000 | A02 | 8000 |

(iii) $\Pi_{Account.balance}$ ($\sigma_{Account.balance < d.balance}$ (Account × $\rho_d$ (Account)))

| Account.balance |
|---|
| 7000 |
| 6000 |

iv) $\Pi_{balance}(Account) - \Pi_{Account.balance}\left(\sigma_{Acc.balance < d.balance}\left(Account \times \left(\rho_d(Account)\right)\right)\right)$

| balance |
|---|
| 8000 |

Example:-

Find the names of all customers who live on the same street and same city as 'smith'.

⇒

$$\Pi_{customer.customer\_name}\left(\sigma_{customer.customer\_city = smith\_addr.city \wedge customer.customer\_street = smith\_addr.street}\left(Customer \times \left(\rho_{smith\_addr(city,street)}\left(\Pi_{customer\_city,\ customer\_street}\left(\sigma_{customer\_name = 'smith'}(customer)\right)\right)\right)\right)\right)$$

Process:

Let customer table has following instance.

Customer

| Customer_name | Customer_city | Customer_street |
|---|---|---|
| Ram | Ktm | New Road |
| Smith | Ktm | New Road |
| Shyam | Lalitpur | Ring Road |

i)

| Customer_name | Customer_city | Customer_street |
|---|---|---|
| Smith | ktm | New Road |

ii)

| Customer_city | Customer_street |
|---|---|
| ktm | New Road |

iii) $\Pi_{smith\_addr(city,street)}$

| City | Street |
|---|---|
| ktm | NRw Road |

iv)

| customer | customer_city | customer_street | City | Street |
|---|---|---|---|---|
| Ram | ktm | New Road | ktm | NR |
| smith | ktm | New Road | ktm | NR |
| Shyam | Lalitpur | Ring Road | ktm | NR |

v)

| Customer_name | customer_city | customer_street | city | street |
|---|---|---|---|---|
| Ram | ktm | NR | ktm | NR |
| Smith | ktm | NR | ktm | NR |

vi)

| customer_name |
|---|
| Ram |
| Smith |

## Formal Definition of Relational algebra:-

The fundamental operations of relational algebra allow us to give a complete definition of an expression in the relational algebra. It can be defined as follows:

1) A basic expression in the relational algebra consists of one of the following:
   * A relation in the database
   * A constant relation

   A constant relation is written by listing its tuples within { }, for example, {A-101, Kathmandu, 5000}

2) A general expression in relational algebra is constructed out of smaller sub expressions. Let $E_1$ and $E_2$ be the relational algebra expressions. Then, these are all relational algebra expressions.
   a) $E_1 \cup E_2$
   b) $E_1 - E_2$
   c) $E_1 \times E_2$
   d) $\sigma_p(E_1)$ where 'p' is the predicate on attributes in $E_1$
   e) $\Pi_s(E_1)$, where 's' is the list consisting of some of the attributes in $E_1$
   f) $\rho_x(E_1)$, where x is the new name for the result of $E_1$.

## Additional Relational Algebra Operations :-

- The fundamental operations of the relational algebra are enough to express any relational algebra query.
- But some common queries are lengthy to express if we only use fundamental operations.
- Additional operations such as set intersection, natural joint, division and assignment operation simplified the common queries.

### 1) Set Intersection operation :

Let r and s are two relations having same arity and attributes of r and s have compatible domain, then their intersection $r \cap s$ is defined as follows:

$$r \cap s = \{t \mid t \in r \text{ and } t \in s\}$$

Set intersection operation can be determined by using fundamental operation, set difference as below:

$$r \cap s = r - (r - s)$$

**Example:-**

Let r and s are two relations:

| A | B |
|---|---|
| α | 1 |
| α | 2 |
| β | 1 |

relation r

| A | B |
|---|---|
| α | 2 |
| β | 3 |

relation s

$r \cap s$:

| A | B |
|---|---|
| α | 2 |

Example:-

Find the names of all customers who have both loan and account in a bank.

→

$$\Pi_{customer\_name}(Deposits) \cap \Pi_{customer\_name}(Borrows)$$

II) Natural Join Operation:

- Natural Join is a bingary operation and is the extension of cartesian product operation.
- It combines cartesian product, selection and projection operation into a single operation.
- It removes duplicates from joined relation and perform equality test of common attributes automatically.
- It is denoted by the symbol ⋈.
- Formally, let r and s are two relations on schema R and S respectively then their natural join is denoted by $r \bowtie s$ and is defined as follows:

$$r \bowtie s = \Pi_{R \cup S}\left(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n}(r \times s)\right)$$

where, $A_1, A_2, \ldots, A_n$ are common attributes of R and S.

Example:-

| A | B | C |
|---|---|---|
| α | 1 | a |
| β | 2 | b |
| γ | 3 | c |

relation r

| B | C | D |
|---|---|---|
| 1 | a | α |
| 2 | a | β |
| 2 | b | γ |

relation s

r ⋈ s:

| A | B | C | D |
|---|---|---|---|
| α | 1 | a | α |
| β | 2 | b | γ |

Example:- Find the names of all customer who have loan in the bank.

$$\Pi_{customer\_name}(Borrows \bowtie Customer)$$

Example:- Find the names of all customer who have account in the bank and stay in ktm.

$$\Pi_{customer\_name}(\sigma_{customer\_city='ktm'}(Deposites \bowtie Customer))$$

## Theta Join :-

- The theta join operation is an extension to the natural join operation that allows us to combine a selection and a cartesian product into a single operation.
- Consider relation r and s, and let $\theta$ be a predicate on attribute in the schema R U S.
- Then, the theta join operation $r \bowtie_\theta s$ is defined as follows:-

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

Example:-

Find the names of all customers who have loan at ~~kathmandu branch~~ in bank and stay in kathmandu.

→

$$\Pi_{customer\_name}\left(Customer \bowtie_{customer\_city = kathmandu} loan\right)$$

## III) The Division Operation:-

- The Division operation is denoted by symbol '÷' and is used for solving queries containing "for all" phrase.
- Formally, let r and s be relations on schema R and S respectively. where $S \subseteq R$ (i.e. every attributes of schema S is also in schema R) then $r \div s$ is a relation on schema (R-S) and defined as follows:

$$r \div S = \{t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s (ut \in r)\}$$

Example:-

| A | B |
|---|---|
| $S_1$ | $P_1$ |
| $S_1$ | $P_4$ |
| $S_2$ | $P_1$ |
| $S_1$ | $P_2$ |
| $S_2$ | $P_4$ |
| $S_3$ | $P_1$ |
| $S_2$ | $P_3$ |
| $S_3$ | $P_2$ |

relation r

| B |
|---|
| $P_1$ |
| $P_2$ |

relation s

$r \div s =$

| A |
|---|
| $S_1$ |
| $S_3$ |

Example:-

Find the names of all customer who have an account in all the bank branch located in Kathmandu.

Step 1:-

$temp1 \leftarrow \Pi_{branch\_name}(\sigma_{branch\_city = 'ktm'}(branch))$

Step 2:-

$temp2 \leftarrow \Pi_{customer\_name, branch\_name}(accounts \bowtie deposites)$

Step 3:-

$\Pi_{customer\_name, branch\_name}(accounts \bowtie deposites) \div \Pi_{branch\_name}(\sigma_{branch\_city = 'ktm'}(branch))$

Result ← temp2 ÷ temp1

iv) Assignment operation:-

- The assignment operation provide convenient way to express complex query.
- It is denoted by ← symbol and works like assignment in programming language.
- By using this operation, a query can be written as a sequential program consisting a series of assignments followed by an expression whose values is displayed as the result of the query.

Example:-

Find all customers who have both loan and account in a bank.

Sol:

temp1 ← Π customer_name (deposits)
temp2 ← Π customer_name (borrows)
result ← temp1 ∩ temp2.

## Extended Relational algebra Expression:-

- The extended relational algebra expression enhance the expressive power of original relational algebra.
- Extended relational algebra operations are listed below:
  1) Generalized projection
  2) Outer Join
  3) Aggregate function.

### 1) Generalized Projection:-

Generalized projection allows arithmetic function in the projection list.

- The general structure is:

$$\Pi_{F1, F2, --, Fn}(E)$$

where,
* E is any relational algebra expression, &
* Each $F_1, F_2, --, F_n$ are arithmetic expression involving constraints & attributes in the schema of E.

Example:-

Suppose a relation employee has the following schema:

Employee (employee_name, employee_address, salary)

Query :- Find the employee and their corresponding bonus, assume that bonus for each employee is 10% of his/her salary.

$\Pi_{employee\_name,\ (salary * 0.1)\ as\ bonus}(employee)$

OR

$Result(name, bonus)(\Pi_{employee\_name,\ salary * 0.1}(employee))$

2) Outer Join Operation:-
- The outer join is an extension to natural join.
- It is used to display rows in the result that do not have matching values in the join column.
- There are three types of outer join operations:
  i) Left-outer join ( ⟕ )
  ii) Right-outer join ( ⟖ )
  iii) Full outer join ( ⟗ )

1) Left-outer join (⟕):-
- The left outer join of two relation R and S is a join in which matching tuples from both relations as well as tuples from R (left relation) that do not have matching values in common columns of S are also included in result relation.
- For unmatched tuples attributes from relation S

right relation are padded with NULL values.

Example:- Let us consider the relation customer and deposits have following instances:

customer

| customer_name | customer_city | customer-street |
|---|---|---|
| Rohan | ktm | New Road |
| Hari | pkr | Buddha chowk |
| Shyam | tkp | Siddhartha |

deposites

| customer_name | account_number |
|---|---|
| Rohan | A-01 |
| Hari | A-02 |
| Govinda | A-03 |

customer ⋈ deposites

| customer_name | customer_city | customer_street | account_number |
|---|---|---|---|
| Rohan | ktm | New Road | A-01 |
| Hari | Pkr | Buddha chowk | A-02 |
| Shyam | Tkp | Siddhartha | NULL |

ii) Right-outer join (⟖):-

- It is a join of two relations R and S in which matching tuples from both relation as well as tuples from S (right relation) that do not have matching values in common columns of R are also included in result relation.
- For the unmatched r tuples, attributes from relation R (left relation) are padded with NULL values.

Example:-

customer ⟖ deposites

| customer_name | customer_city | customer_street | account_number |
|---|---|---|---|
| Rohan | Ktm | New Road | A-01 |
| Hari | Pkr | Buddha Chowk | A-02 |
| Govinda | NULL | NULL | A-03 |

iii) Full-Outer Join (⟗):-

- It is a join of two relations R and S in which matching tuples from both relations as well as tuples from both relations (R and S) that do not have matching values in common columns are also included in result relation.
- For the unmatched tuples from one relation attributes to from another relation are padded with NULL values.

Example:-

customer ⋈ deposites

| customer_name | customer_city | customer_street | account_number |
|---|---|---|---|
| Rohan | ktm | New Road | A-01 |
| Hari | Pkr | Buddha Chowk | A-02 |
| Shyam | TKP | Siddhartha | NULL |
| Govinda | NULL | NULL | A-03 |

3) Aggregate function :-

- Aggregate function takes a collection of values and returns ~~written as a~~ single value as a result.
- Aggregate functions used in database are:
  - i) AVG : Average value
  - ii) MIN : Minimum value
  - iii) MAX : Maximum value
  - iv) COUNT : number of values
  - v) SUM : sum of values.
- The aggregate operation in relational algebra is denoted by the symbol $\mathcal{G}$ (i.e. $\mathcal{G}$ is letter G in calligraphic font)
- The general structure is:

$$G_1 G_2 G_3 \ldots G_n \; \mathcal{G}_{F_1(A_1)\, F_2(A_2), \ldots F_n(A_n)} (E)$$

where,

E is any relational algebra schema.

$G_1, G_2, \ldots G_n$ is a list of attributes on which aggregate function makes group.

undefined ∧ true = undefined
undefined ∧ undefined = undefined
undefined ∧ false = false
undefined ∨ true = true
undefined ∨ false = undefined



Each $F_i$ is aggregate function and each $A_i$ is attribute name.

Example:- Find the total balance of each branch of a bank.

Sol:

$$branch\_name\ \mathcal{G}_{SUM(balance)}\ (account)$$

Example 2:- Find the number of accounts in each branch of a bank.

Sol:

$$branch\_name\ \mathcal{G}_{COUNT(account)}\ (account)$$

## NULL Values:-

NULL values represents the values for an attribute that is currently unknown or not applicable for the tuple. It deals with incomplete or exceptional data. It also represents the absence of value and is not same as 0 or spaces because they are values. Different operations deals with NULL value which are as follows:

### 1) Arithmetic and comparison operation:-

Arithmetic operations such as +, −, *, / involving NULL values must return NULL values. Similarly, any comparison operation such as <, >, <=, >= and ≠ involving a NULL value evaluating a special value unknown so we cannot say for sure

the result of the comparison is true or false.

ii) **Boolean operation :-**

Boolean operation deals with NULL values as follows:

a) OR operation

unknown OR unknown = unknown
unknown OR true = true
unknown OR false = unknown

b) AND operation

unknown AND unknown = unknown
unknown AND false = false
unknown AND true = unknown

c) NOT operation

NOT (unknown) = unknown

iii) **Select operation :-**

If predicate returns true value then tuple is added to the result otherwise predicate returns NULL or false then tuple is not added to the result.

Predicate (E)

iv) Join operation :-

Join can be expressed as a cross product followed by a selection. Thus, the definition of how selection handles NULL also defines how join operations handle NULL.

v) Projection :-

The projection operation treats NULL values just like any other values when eliminating duplicates.

vi) Union, Intersection and Difference :-

These operations treat NULL values just as the projection does.

vii) Generalized projection :-

Duplicate tuple containing NULL values are handled as in the projection operation.

viii) Aggregate :-

When NULL values occur in grouping attributes, the aggregate operation treats them just as projection.

ix) Outer Join :-

Outer Join operation behaves just like join operations except on tuples that do not occur in the join result.

## Modification of Database :-

Insertion, Deletion and Update operations are responsible for database modification.

### 1) Insertion operation :-

To insert data into a relation, we either specify a constant tuple or by writing the queries whose result is a set of tuples to be inserted. We need to use assignment and union operations to insert new tuples into a relation.

General form of insertion operation is given below:

$$r \leftarrow r \cup E$$

where,

r is a relation &
E is a constant tuple or a relational algebra expression.

- Since we are using union operation, result of E must be union compatible with relation r.

Example: Insert a information in the database specifying customer name 'X' having balance 5000 and account number A-900 at the Kathmandu branch.

Soln:

account ← account ∪ {'A-900', 'ktm', 5000}
deposites ← deposites ∪ {'X', 'A-900'}

Example2: Query:-
Suppose a bank want to provide as a gift for all loan customers of the Kathmandu branch a new $1000 saving account.

Sol:

$r_1 \leftarrow \sigma_{branch\_name = 'ktm'} (borrows \bowtie loan)$
$r_2 \leftarrow \Pi_{loan\_number, branch\_name} (r_1)$
$accounts \leftarrow accounts \cup (r_2 \times \{(1000)\})$
$deposites \leftarrow deposites \cup \Pi_{customer\_name, loan\_number} (r_1)$

## ii) Delete operation:-

This operation is used to delete information from a relation. We need to use assignment and set difference operation to delete tuples from a relation. To use delete operation, we must be careful about union compatibility property holds between relations, because we are using set difference operation.
General form of delete operation is given below:

$$r \leftarrow r - E$$

where,
r is a relation &
E is a constant or a relational algebra expression

Examples:-

1. Delete all of Smith's account record.

Sol:

$deposites \leftarrow deposites - (\sigma_{customer\_name = 'smith'} (deposites))$

2. Delete all loan with amount in the range 0-50.

Sol:

$$loan \leftarrow loan - (\sigma_{0 \leq amount \wedge amount \leq 50}(loan))$$

3. Delete all acounts from the branch located in Kathmandu.

Sol:

$$r_1 \leftarrow \sigma_{branch\_city = 'ktm'}(branch \bowtie account)$$
$$r_2 \leftarrow \Pi_{account\_number, branch\_name, balance}(r_1)$$
$$account \leftarrow account - r_2$$

iii) Update operation:-

Update operation is used to change an attribute value in a tuple without changing rest of the values in that tuple.

- Generalized projection is used to perform update operation.
- If we use only generalized projection without specifying any condition, it will update the specified attribute value in all tuples.
- It is expressed as follows:-

$$r \leftarrow \Pi_{F_1, F_2, F_3, \ldots, F_n}(r)$$

where,

r is a relation &
$F_i$ is an arithmetic expression or an attributes.

- If we want to select some tuples from r and to update only them, we can use the following expression:

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(\sigma_P(r)) \cup r - \sigma_P(r))$$

where,

P denotes the selection condition that chooses which tuples to update.

Example:-

1. Increase all balance of a bank by 5%.

Sol^n:

$$accounts \leftarrow \Pi_{account\_number,\ branch\_name,\ balance * 1.05}(account).$$

2. Increase balance by 6% for those accounts whose balance is greater than 10,000 and for rest of accounts increase balance by 5%.

Sol^n:

$$accounts \leftarrow \Pi_{account\_number,\ branch\_name,\ balance * 1.06}(\sigma_{balance > 10,000}(account)) \cup \Pi_{account\_number,\ branch\_name,\ balance * 1.05}(\sigma_{balance \leq 10,000}(account))$$

3. Consider the relational database of given schema, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries.

Employee (person_name, street, city)
works (person_name, company_name, salary)
company (company_name, city)
manages (person_name, manager_name)

a) Find the name of all employees who works for First Bank Corporation.

Sol^n:

$\Pi_{person\_name}(\sigma_{company\_name = "First\ Bank\ corporation"}(works))$

b) Find the names and cities of residence of all employees who work for First Bank Corporation.

Sol^n:

$\Pi_{person\_name, city}(\sigma_{company\_name = 'First\ Bank\ corporation'}(employee \bowtie works))$

c) Find the name, city and street of residence of all employees who work for First Bank corporation and earn more than $10,000 per annum.

Sol^n:

$\Pi_{person\_name, city, street}(\sigma_{company\_name = 'FBC' \wedge salary > 10,000}(Employee \bowtie works))$

d) Find the names of all employees who live in the same city as the company for which they work.

Sol^n:

$\Pi_{person\_name}(employee \bowtie works \bowtie company)$

e) Find the names of all employees who live in the same city and on the same street as do their manager.

Sol^n:

$\Pi_{person\_name}((employee \bowtie manages) \bowtie_{manages.manager\_name = employee2.person\_name \wedge employee.city = employee2.city \wedge employee.street = employee2.street} (\rho_{employee2}(employee)))$

f) Find the names of all employees in the database who do not work for first bank corporation.

Sol:

$$\Pi_{person\_name}(\sigma_{company\_name \neq 'FBC'}(employee \bowtie works))$$

OR

$$\Pi_{person\_name}(works) - \Pi_{person\_name}(\sigma_{company\_name = "FBC"}(works))$$

g) Find the names of all employees who earn more than ~~no~~ every employee of First Bank Corporation.

Sol:

$$\Pi_{person\_name}(works) - \Pi_{person\_name}(works \bowtie_{works.company\_name = "FBC" \wedge works.salary < works2.salary} \rho_{works2}(works))$$

h) Assume the companies may be located in several cities. Find all companies located in every city in which small bank corporation is located.

Sol:

$$\Pi_{company\_name, city}(company) \div \Pi_{city}(\sigma_{company\_name = "SBC"}(company))$$

Q: Assume a database about company
Employee (ss#, name)
Company (cname, address)
works (ss#, cname)
supervises (supervisor_ss#, employee_ss#)
Write relational algebra queries for each of the following cases.

a) Find the names of supervisors that work in companies whose address equal Biratnagar.

$$\Pi_{name}\left(\sigma_{\substack{supervises.supervisor.ss = employee.ss \\ \wedge employee.ss = works.ss \\ \wedge company\_cname = works.cname \\ \wedge company.company\_address = \text{"Biratnagar"}}}\left(\text{Employee} \bowtie \text{Company} \bowtie \text{works} \bowtie \text{supervises}\right)\right)$$

b) Find the names of an companies who have more than 10 employees.

$$cname\ \mathcal{G}_{COUNT(employee.ss) > 10}\left(\text{Employee} \bowtie \text{company} \bowtie \text{works}\right)$$

c) Find the name of supervisor who has minimum number of employees.

## Structured Query Language (SQL):-

- SQL stands for Structured Query Language.
- It is used to communicate with database.
- It is designed for the retrieval and management of data in relational database management system.
- It is also used for schema creation and modification and database access control management.
- It consists of following parts:

### i) Data Definition Language (DDL):-

It provides commands for defining, deleting and modifying relations and view schemas. It also includes commands for ~~integ~~ specifying integrity constraints that the data stored in the database must satisfy and commands for specifying access rights to relations and views.

### ii) Data Manipulation Language (DML):-

The SQL DML includes query language based on both relational algebra and the tuple relational calculus. It also includes commands to insert the tuple into, delete tuple from and modify tuple in the database.

### iii) Embedded and Dynamic SQL:-

Embedded SQL defines how SQL statements can be embedded within general purpose programming

languages such as C, C++, Java, etc.
Dynamic SQL allows us to construct queries at run time.

iv) Transaction Control:-
SQL includes commands for specifying the beginning and ending of transactions.

v) View Definition:-
The SQL DDL includes commands for defining views.

vi) Integrity:-
The SQL DDL includes commands for specifying integrity contraints that the data stored in the database must satisfy.

vii) Authorization:-
The SQL DDL includes commands for specifying access rights to relations and views.

## Data Definition Language (DDL):

- The DDL is a part of SQL, it allows creation, deletion and modification of database object.
- It also allows to define index (key) which specify links between tables and impose constraints between database tables.
- The most important DDL statements in SQL are

i) CREATE statement:
Used to create a table, view or other database objects.

ii) ALTER statement:
Used to alter (modify) database table, view or other database objects.

iii) DROP statement:
Used to delete database tables, views or other database objects.

(Domains)

## Basic Data Types in SQL:

The SQL standard supports the variety of built-in data types including

i) char(n) or character(n):-
A fixed length string with specified length.

ii) varchar(n) :-
A variable length character string with user specified or defined maximum length.

iii) int or integer :-
A finite set of integers that is machine dependent.

iv) small int :-
A small integer.

v) Numeric (p,d) :-
A fixed point number with user specified precision. They have fixed no. of digits to the left of the decimal point and a fixed number of digits to right. The total number of digits on both sides of the decimal are precision. The number consists of p digits and d of the p digits is to the right of the decimal point.

vi) Real, Double precision :-
These are approximate numeric values that have a fixed precision (number of digits) but a floating decimal point.

vii) Float(n):-
A floating-point number with precision of at least n digits.

viii) Date:-
calendar date with year, month and day.
eg:- 2009-02-12

ix) Time:-
clock time with hour, minute and second. It may also contain fraction of second.
eg:- 11:30:42

x) Timestamp:-
Combination of calendar date and clock time with year, month, day, hour, minute, second and fraction of second.
eg:- '2009-02-12 11:30:42:45'

xi) Interval:-
It represents time and date intervals.

## CREATE statement:-

The CREATE statement is used to create a different structures of the database such as tables, views, triggers, indexes, etc.

## CREATE TABLE (Table):-

The main features of tables are:

- It is a relation that is used to store records of related data.
- It is a logical structure maintained by the database manager.
- It is made up of columns and rows.
- At the intersection of every rows and columns there is a specific data item called value.
- A base table is created with the ~~creat~~ CREATE TABLE statement and is used to hold persistent user data.

Syntax:-

```
CREATE TABLE <table name>
(
<column_name1> <data_type> [NOT NULL] [UNIQUE]
                    [INTEGRITY_CONSTRAINT],
<column_name2> <data_type> [NOT NULL] [UNIQUE]
                    [INTEGRITY_CONSTRAINT],
---------- ---- ---- ---- ---- ---- ----
<column_namen> <data_type> [NOT NULL] [UNIQUE]
                    [INTEGRITY_CONSTRAINT]
);
```

Example:-

```
CREATE TABLE statement student
(
 Sid INTEGER NOT NULL,
 Sname VARCHAR (12),
 level VARCHAR (12),
 age INTEGER,
 gender VARCHAR (6),
 PRIMARY KEY (sid)
);
```

Output of the above statement is:

| Sid | Sname | level | age | gender |
|---|---|---|---|---|
| | | | | |
| | | | | |

## ALTER TABLE statement:-

- It allows us to modify the given table.
- The structure of the given table can be changed either of the following ways:

i) By adding new column in existing table.
ii) By deleting some columns from an existing table.
iii) By modifying some columns of given table.

1) Adding column to the existing table :-

Syntax:-

```
ALTER TABLE <table_name>
ADD (<column_name> <data_type>);
```

Example:-

Suppose we want to add a new column 'address' to an existing table Student.

```
ALTER TABLE student
ADD (address VARCHAR(12));
```

If we execute the above query then we get the following table:

| sid | Sname | level | age | gender | address |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

ii) Removing column from existing table:-

Syntax:-

```
ALTER TABLE <table_name>
DROP (column_name);
```

Example:-

Suppose we want to remove existing column 'address' from the student table, then

```
ALTER TABLE student
DROP (address);
```

iii)

If we execute the above query then we get the following table:

| Sid | Sname | level | age | gender |
|---|---|---|---|---|
| | | | | |
| | | | | |

iii) Modifying an existing column of table :-

Syntax:-

ALTER TABLE <table_name>
MODIFY (<column_name>, <data_type>)

Example:-

Modify student relation by changing the range of the name of student by 30.

→ - ALTER TABLE student
MODIFY (sname, VARCHAR(30))

DROP TABLE statement:-

It allows us to remove an existing table from the database.

Syntax:-

DROP TABLE <table_name>

Example:- If we want to remove a table statement from the database, then

DROP TABLE student.

## Example schema:-

All queries provided in this chapter are based on the following relational schema:

Branch_schema = (branch-name, branch-city, assets)
Customer_schema = (customer_name, customer-city, customer-str
Loan_schema = (loan_number, branch_name, amount)
Borrows_schema = (customer_name, loan_number)
Account_schema = (account_number, branch_name, balance)
Deposites_schema = (customer_name, account_number).

## Basic Structure of SQL Query:-

Basic structure of SQL expression consists of three clauses : ~~select~~ SELECT, FROM and WHERE.

- The SELECT clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of query.

- The FROM clause corresponds to the Cartesian product operation of the relational algebra. It list the relations to be scanned in the evaluation of the expression.

The WHERE clause corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in

the FROM clause.

A typical SQL query has following form:

SELECT $A_1, A_2, \ldots A_n$
FROM $r_1, r_2, r_3, \ldots, r_n$
WHERE P

where,

- each $A_i$ represents an attribute and each $r_i$ a relation.
- P is a predicate.

→ The equivalent query of relational algebra expression is

$$\Pi_{A_1, A_2, A_3, \ldots, A_n}(\sigma_P (r_1 \times r_2 \times r_3 \times \ldots \times r_n))$$

## The SELECT clause:-

The SELECT clause specifies a list of columns to be retrieved from the tables in the FROM clause. It has following general format:

SELECT [ALL | DISTINCT] select_list

- select_list is a list of column names separated by commas.
- The ALL | DISTINCT specifiers are optional.

The ALL specifies selection of all row.
The DISTINCT specifies that duplicate rows are eliminated.
The default value is 'ALL'.

Example:-
Find the names of all branches in the loan relation.

→ SELECT branch_name
FROM loan

The FROM clause:-
The FROM clause always follows the select clause. It lists the tables accessed by the query. When the FROM list contains multiple tables, then commas are used to separate the table names. It is equivalent to cartesian product of relational algebra.

Syntax:-
SELECT [ALL|DISTINCT] select_list
FROM table_name.

Example:-
Find the ~~loan~~ names, loan_number and loan_amount of all customer who have loan at the bank.

SELECT customer_name, loan_number, amount
FROM ~~customer~~ borrows, loan
WHERE ~~customer~~ borrows.~~customer_name~~ loan_number = loan.~~customer_name~~ loan_number

## The WHERE clause:-

It is used to specify a condition while fetching a data from a single table or by joining with multiple tables. It is an optional clause. When specified, it always follows the FROM clause.

Syntax:

```
SELECT select_list
FROM table_name
WHERE condition
```

Following are the comparion expressions that compares the content of ~~column~~ table column to a literal. It may also compares two columns to each other.

i) = equals
ii) > greater than
iii) < less than
iv) >= greater than or equal to
v) <= less than or equal to
vi) <> not equal to

Example: Find all loan numbers for loans made at the Kathmandu branch with loan amounts greater than $12,000.

→
```
SELECT loan_number
FROM loan
WHERE branch_name = 'Ktm' AND amount>12,000
```

Example 2:
Find the loan number of those loans with loan amounts between $90,000 and $1,00,000.

→ SELECT loan_number
FROM loan
WHERE amount >= 90,000 AND amount <= 100000
or
WHERE amount BETWEEN 90000 AND 100000.

## The Rename Operation:-

SQL provides a mechanism for renaming both relations and attributes. It has following form:
<old_name> AS <new_name>

The AS clause appears in both SELECT and FROM clause.

Example: Consider the following query.

```
SELECT customer_name, borows.loan_number, amount
FROM loan, borrows
WHERE loan.loan_number = borrows.loan_number
```

The result of this query is a relation with the following attributes.
customer_name, loan_number, amount.

If we want the attribute name loan_number to be replaced with the name loan_id, we can rewrite

the query as:

```
SELECT customer_name, borrows.loan_number AS loan_id,
       amount
FROM loan, borrows
WHERE loan.loan_number = borrows.loan_number.
```

## String Operation (Pattern Matching):-

Pattern Matching is one of important operation with string. SQL allows pattern matching with string by using 'LIKE' operator. We describe pattern matching by using two special characters.

1) Percent (%):- The % character matches any substring.
2) Underscore (_):- The _ character matches any single character.

- Patterns are case sensitives i.e. uppercase character do not match with lowercase character or vice-versa.
- Following example illustrates the pattern matching.

i) 'Bi%' :- Matches strings that start with Bi.
ii) '%sh' :- Matches all strings that ends with sh.
iii) '%rj%' :- Matches all strings that contains the substring rj anywhere.
iv) '___' :- Matches any strings of exactly 3 character.
v) '___%' :- Matches any strings of at least 3 characters.

## General structure of LIKE operator :-

match_expression [NOT] LIKE pattern [ESCAPE escape_character]

where,

i) match_expression :- It is any valid expression of character data type.

ii) pattern :- It is the specific character to search for in match expression.

iii) escape_character :- It is a character that is put infront of a wild card character to indicate that the wild card should be interpreted as a regular character and not as wild card. The escape character must be a string containing single character.

Example:-

Find the names of all customers whose address contains 'main' as a substring.

```
SELECT customer_name
FROM customer
WHERE customer_street LIKE '%main%'
```

Example 2: Find the names of all customer whose name contains underscore.

→

```
SELECT customer_name
FROM customer
WHERE customer_name LIKE '%\_%' ESCAPE '\';
```

## Ordering the Display of Tuples:

The 'ORDER BY' clause is used to change the display ordering of tuples. It is an optional clause. It has following general format:

```
ORDER BY column_name [ASC|DESC]
```

where,

column_name is an attribute in which we want to apply the ORDER BY clause and ASC and DESC request ascending and descending sort of a column. ASC is a default value.

Example:- Find the names of all customer in alphabetical order, and who lives in Kathmandu.

→

```
SELECT customer_name
FROM customer
WHERE customer_city = 'ktm'
ORDER BY customer_name
```

Example 2: Display the loan table in descending order of loan amount and if more than one loan has the same amount then ascending order of loan number.

→

```
SELECT *
FROM loan
ORDER BY amount DESC, loan_number ASC
```

## Set Operations :

- The set operations UNION, INTERSECTION and EXCEPT allow us to combine two or more select statements.

### 1. Union :-

- It combines two or more result sets into a single set without duplication.
- If we want to retain duplicates we write UNION ALL in place of UNION.

Example: Find all customers who have loan, account or both at the bank.

Sol:

```
( SELECT customer_name
   FROM deposits)
     UNION
( SELECT customer_name
   FROM borrows)
```

### 2. Intersection:-

- Takes the data from both result set which are in common
- If we want to retain all duplicates, we must write INTERSECT ALL in place of INTERSECT.

Example: Find all customers who have both loan and account at the bank.

Sol:

```
(SELECT customer_name
 FROM deposits)
  INTERSECT
(SELECT customer_name
 FROM borrows)
```

3. Except:-

- Takes the data from first result set but not in the second.
- If we want to retain all duplicates we must write EXCEPT ALL in place of EXCEPT.

Example: Find all customers who have account but no loan at the bank.

Sol:

```
(SELECT customer_name
 FROM depositors)
  EXCEPT
(SELECT customer_name
 FROM borrows)
```

## Aggregate Functions:-

Aggregate functions are those functions that take a collection of values as input and return a single value. SQL offers 5 built-in functions:

1. Average : AVG
2. Minimum : MIN
3. Maximum : MAX
4. Total : SUM
5. Count (no of selected column) : COUNT

Example:- Find the average account balance at the kathmandu branch.

Sol:

```
SELECT AVG (balance)
FROM account
WHERE branch_name = "kathmandu"
```

Example:- Find the average account balance at each branch.

Sol:

```
SELECT AVG (balance)
FROM account
GROUP BY branch_name
```

arko query vitra lekhinxa - sub-query



Example :- Find the number of depositors for each branch.

Sol:

```
SELECT branch_name, COUNT (account_number)
FROM account
GROUP BY branch_name.
```

Example: Find those branches whose average account balance is 50,000.

Sol:

```
SELECT branch_name, AVG (balance)
FROM account
GROUP BY branch_name
HAVING AVG (balance) = 50000
```

## Nested Subquery :-

SQL provides a mechanism for nesting sub-queries. A subquery is a SELECT_FROM_WHERE expression that is nested within another query. A common use of subquery is to perform test for set membership, set comparison and empty relation.

### 1. Set membership :-

SQL has some operations that allows testing tuples for membership in a relation. The IN connective test for set membership where the set is a

collection of values produced by SELECT clause. The NOT IN connective test for absence of the set membership. The IN connective has following general format:

val1 [NOT] IN {val2, val3, ... valn}

This comparison test if val1 matches val2 or val3 or val4 and so on. It is equivalent to following logical expression:

val1 IN {val2, val3, ..., valn} = {val1 = val2 OR val2 = val3 OR .... val1 = valn}

Example: Find all customers who have both loan and account.

Sol:

```
SELECT customer_name
FROM deposits
WHERE customer_name IN ( SELECT customer_name
                         FROM borrows)
```

2. Set Comparison :-

The nested subquery can be used to compare the sets. The key word ALL or ANY can be used for set comparison. The SOME is an equivalent keyword for ANY. ANY and SOME must matches at least one row in the subquery. ALL must matches all rows in the subquery.

Example: Find the names of all branches assets greater than those of atleast one branch located in Pokhara.

Soln:

```
SELECT branch_name
FROM branch
WHERE assets>SOME ( SELECT assets
                    FROM branch
                    WHERE branch_name = 'Pokhara')
```

3. Test for empty Relation:

SQL includes a feature for testing whether a subquery has any tuples in its result. The EXIST construct returns the value true if arguments of query is non-empty. We can test for non-existence of the tuples in a query by using the NOT. EXIST construct.

Example: Find all customers who have loan and account at the bank.

Soln:

```
SELECT customer_name
FROM borrows
WHERE EXISTS ( SELECT *
               FROM deposits
               WHERE
               borrows.customer_name
               = deposits.customer_name)
```

retrieve garda step by step garnu pardaina, direct garna sakinxa, non-procedural query language. 

## Relational calculus:

Relational calculus is a non-procedural query language. i.e. the user is not concerned with the detail of how to obtain the end result. In relational calculus, a query is expressed as a formula consisting variables. It is mainly based on well-known propositional calculus. In relational calculus, the sentences we deal are simpler and refer specifically to relations and values in the database of interest. Relational calculus is of two types:

1) Tuple Relational calculus &
2) Domain Relational calculus.

## 1. Tuple Relational Calculus (TRC):-

Tuple Relational calculus is a logical language with variable ranging over tuples. The general form of tuple relational calculus is given by:-

$\{t \mid COND(t)\}$

Here → t is the tuple variable which stands for tuple of relation.

→ COND(t) is a formula that describes t.

Example: Find the names of all customers in a bank.

Sol^n:

$\{t.customer\_name \mid customer(t)\}$

## Quantifiers:-

Quantifiers are words that refer to quantities. Quantifiers can be broadly classified into two types:

i) Universal quantifier
ii) Existential quantifier

### i) Universal Quantifier:-

- The expression $\forall x\, P(x)$ represents universal quantification of $P(x)$.
- It means "for all x, $P(x)$ is true."

### ii) Existential Quantifier:-

- The expression $\exists x\, P(x)$ denotes the existential quantification of $P(x)$.
- It means "there is at least one x such that $P(x)$ is true."

tuple hold garne capability vayeko variable
row

## Tuple variable:-

A variable that can hold tuple of a relation is called tuple variable. It is of two types:

i) Free-tuple variable
ii) Bounded tuple variable

Any variable that is not bounded by quantifier is set to be free.

Any variable that is bounded by quantifier is set to be bounded variable.

Example:- Consider the following relational calculus query.
$\{s.rollNO, S.Name \mid student(s) \wedge s.sex = 'F' \wedge (\exists d)$
$(department(d) \wedge d.name = 'Math' \wedge d.deptId = s.deptNo)\}$

Here,
s = free variable
d = existentially bound tuple variable

## Formal Definition of TRC Formulae:-

- A TRC expression is of the form $\{t \mid P(t)\}$ where P is a formula.
- Several tuple variables appear in a formula.
- A variable may be free or bound in a formula.
- A TRC formula is built up out of automic formula.
- An automic formula has one of the following forms:

i) $t \in r$ :- where t is a tuple variable & r is a specified relation.

ii) $t.a$ op const :- t is a tuple variable, a is an attribute on which t is defined, op is a comparison operator $(<, >, =, \neq, \geq, \leq)$ & const is a constant in the domain of attribute a.

wff - well formed formula.

AOM → right side either attribute or constant.
↳ left side always attribute

(140)

iii) t.a op s.b :- s and t are tuple variables, a is attribute on which t is defined, b is attribute on which s is defined, op is a comparison operator.

- The general formula is built up from atoms using the following:

i) An atomic formula is a formula.
ii) If P is a formula, then so are ¬P and (P).
iii) If $P_1$ and $P_2$ are formulae, then so are $P_1 \wedge P_2$, $P_1 \vee P_2$ and $P_1 \Rightarrow P_2$
iv) If P(t) is a formula containing a free variable t then $\exists t \in r\,(P(t))$ and $\forall t \in r\,(P(t))$ are also formulae.

**Well-formed formula:-**

Valid form of logical expression involving tuple variable are called well-formed formula and are defined as follows:

1. AθM is a wff. If A is a projection of a tuple variable, M is a constant or a projection of tuple variable and θ is the one of the comparison operators: $=, \neq, <, >, \leq, \geq$.

2. $F_1$ & $F_2$ and $F_1 | F_2$ are wff if $F_1$ and $F_2$ are wff.

3. (F) is a wff if F is a wff.

different data bata information select garda criteria set gardinxa.



4. $\exists x\,(F(x))$ and $\forall x\,[F(x)]$ are wff if $F(x)$ is a wff with a free occurance of a variable $x$.

## Safety of Expression:

A Tuple Relational Calculus (TRC) expression may generate an infinite relation. Suppose that we have write an expression:

$$\{t \mid \neg(loan(t))\}$$

There are infinitely many tuples that are not in loan. Most of those tuples values that do not even appear in the database. To solve the above problem the concept of domain of a tuple relational formula P is defined. The domain of P is denoted by DOM(P). It is the set of all values referred by P. The domain of P is the set of all values that appear explicitly in P or that appear in one or more relations whose names appear in P.

Example:

- $DOM(loan(t) \wedge t.amount > 1200)$ is the set containing 1200 as well as the set of all values appearing in loan.
- $DOM(\neg(loan(t)))$ is the set of all value appearing in loan, since the relation loan is mentioned in the expression.

Example Queries:

1) Example 1: Find the branch_name, loan_number and amount for loans over $1200.

Sol:

$\{ t \mid loan(t) \wedge t.amount > 1200 \}$

2) Example 2: Find the names of all customers who have loan from Kathmandu branch.

Sol:

$\{ t.customer\_name \mid \exists s \in borrows (t.customer\_name = s.customer\_name \wedge \exists u \in loan (u.loan\_number = s.loan\_number \wedge u.branch\_name = "ktm")) \}$

## Limitations of TRC:

1. Tuple Relational Calculus cannot express queries involving aggregations, groupings and ordering.
2. It also cannot calculate transitive closure.
3. For eg: we cannot count no. of employees in particular department and we also cannot order the display of data according to some column.

find out garne lai left side ma, aru baaki right ma



## 2. Domain Relational Calculus (DRC):-

Domain Relational calculus is non-procedural query language. It is a form of relational calculus which uses domain variables that takes on values from an attributes domain rather than values for an entire tuple.

In domain relational calculus, each query is an expression of the form

$\{<x_1, x_2, x_3, \ldots, x_n> \mid P(x_1, x_2, x_3, \ldots, x_n)\}$

where $x_1, x_2, x_3, \ldots, x_n$ represents domain variable & P represents a formula.

Example 1:

Find branch name, loan number & amount for loans of over $1200.

Soln:

$\{<l, b, a> \mid <l, b, a> \in loan \wedge a > 1200\}$

Example 2:

Find the names of all customers who have a loan from the kathmandu branch and find the loan amount.

Soln:

$\{<n, a> \mid \exists_l (<n, l> \in borrows \wedge \exists_b (<l, b, a> \in loan \wedge b = \text{"kathmandu"}))\}$

## Query By Example (QBE):

- QBE stands for Query By Example.
- It is a data manipulation language and early database system includes this language.
- It was developed by IBM in early 1970. Today many database system for personal computer support variants of QBE language.
- QBE has two distinctive features:

a) Unlike most query languages and programming language, QBE has 2-dimensional syntax i.e. queries look like tables.

b) QBE queries are expressed by examples. Instead of giving a procedure for obtaining the desired answer, the user gives an example of what is desired. The system generalizes this example to compute the answer to the query.

## QBE Query Example:

Example 1: Projection Operation

Find all customer names whose address is Kathmandu.

Sol:

| customer | customer_name | customer_city | customer_street |
|---|---|---|---|
| | P. | Kathmandu | |

Here,

P. implies print. To make a projection only put P. in any column of the relation.

Example 2: Selection Operation

To make a selection, put quantities in the column of the attributes in the question. To print whole records, put P in the column with the name of the record.

Q Print all information of the customer who live in kathmandu.

Sol:

| customer | customer_name | customer_city | customer_street |
|---|---|---|---|
| P. | | kathmandu. | |

Example 3: AND Condition.

Find the loan numbers of all loans jointly made by Smith and John.

Sol:

| borrows | loan_number | customer_name |
|---|---|---|
| | P._x | "Smith" |
| | _x | "John" |

Example 4: OR condition

Print all the account number from the kathmandu or Ghorahi branch and balance is greater than 50,000.

Sol:

| account | account_number | branch_name | balance |
|---|---|---|---|
| P. | | Kathmandu | > 50,000 |
| P. | | Ghorahi | >50000 |

Example 5: Query involving more than one tables

QBE allows queries that span several different relations. The connections among the various relations are achieved through variables that force certain tuples to have the same value on certain attribute such as:

Find the names of all customers who have a loan from the Kathmandu branch.

| loan | loan_number | branch_name | amount |
|---|---|---|---|
| | _x | Kathmandu | |

| borrows | customer_name | loan_number |
|---|---|---|
| | P._y | _x |

Example 6: Ordering of the display of tuples

The records can be arranged either in ascending order or in the descending order using AO and DO operators respectively.

List the customer name in ascending order who have account in a bank.

| deposites | customer_name | account_number |
|---|---|---|
| | P.AO | |

Example 7: Retrieval using negation
The symbol used for negation is ¬.

Find the name of all customers who have an account at the bank but who do not have loan from the bank.

| depositer | customer_name | account_number |
|---|---|---|
| | P._x | |

| borrows | customer_name | loan_number |
|---|---|---|
| ¬ | _x | |

Example 8: Retrieval using condition box
The condition box is used to store logical conditions that are not easily expressed in the tables skeletons.
A condition box can be obtained by pressing a special function key.

Find the loan numbers made by John or Smith or both.

| borrows | customer_name | loan_number |
|---|---|---|
| | _x | P._x |

| conditions |
|---|
| _x = Smith OR _x = John |

## Aggregate Operations:-

QBE includes the aggregate operators AVG, MAX, MIN, SUM and CNT.

i) MIN.ALL implies the computation of minimum value of an attribute.
ii) MAX.ALL implies the computation of maximum value of an attribute.
iii) CNT.ALL implies COUNT the number of tuples in the relations.
iv) SUM.ALL implies the computation of an attribute.
v) AVG.ALL implies the computation of average value of an attribute.

Example:
Find the total balance of all the accounts maintained at the Kathmandu branch.

| account | account_number | branch_name | balance |
|---|---|---|---|
| | | Kathmandu | P.SUM.ALL |

Note: UNQ is used to eliminate duplicates.

Example: Find the total number of customer who have an account at the bank.

| depositor | customer_name | account_number |
|---|---|---|
| | P.CNT.UNQ | |

# Modification of Database:-

## i) Deletion:-

Deletion of tuples from a relation is expressed by use of a D. command.

Example: Delete customer Smith

| Customer | customer_name | customer_street | customer_city |
|---|---|---|---|
| D. | Smith | | |

## ii) Insertion:-

Insertion is done by placing the I. operator in the query expression.

Example: Insert account A-900 at kathmandu branch with balance $70,000.

| account | account_number | branch_name | balance |
|---|---|---|---|
| I. | A-900 | kathmandu | 70,000 |

## iii) Update:-

Use the U. operator to change a value in a tuple without changing all values in the tuple. QBE does not allow users to update the primary key fields.

| ln | [illegible]bn | am[illegible] |
|---|---|---|
| 001 | ktm | 50,000 |
| 002 | Ktm | 60,000 |
| 003 | Pkh | 1,00,000 |

After applying the update operation on loan number 002 the resulting loan table becomes.

| loan | loan_number | bn | amt |
|---|---|---|---|
| U. | 002 | Ghorahi | 60,000 |

| ln | bn | amt |
|---|---|---|
| 001 | ktm | 50,000 |
| 002 | Ghorahi | 60,000 |
| 003 | Pkh | 1,00,000 |

UNIT-2

# Integrity and security

## Integrity constraints :-

Integrity constraints are those condition in database system which guard against invalid database operations or accidental damage to the database by ensuring the authorized changes to the database. It does not allow loss of data consistency in database.

Infact, integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency.

The different types of constraints that can be imposed on the tables are domain constraints, referential constraints, assertions and triggers.

(imp) particular column ma kun value huna kura decide garxa.

### 1. Domain constraints :-

column to attribute ma valid value rakhna banaune

Domain refers to the set of possible values that attributes can take. The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values, a range of values or an expression that accepts the valid values.

For example; (i) the age of the person cannot have any letter from the alphabet. The age should be a numerical value.
(ii) student can have gender either male or female, any other

entry than male or female violates the domain constraints.

- The CREATE DOMAIN clause can be used to define the new domains.
For example; to ensure that age must be an integer in the range 1 to 100, we could use.

```
CREATE DOMAIN ageval INTEGER DEFAULT 0
CHECK (VALUE>=1 AND VALUE <=100)
```

The constraints related to domain constraints are
i) NOT NULL constraints
ii) UNIQUE constraints
iii) PRIMARY KEY constraints
iv) CHECK constraints.

i) NOT NULL constraints:-
If any column is not supposed to take null value then we can impose NOT NULL constraints on that column. The syntax of NOT NULL constraint is:

```
CREATE TABLE <table_name>
( column_name1 data_type_of_column1 NOT NULL
  column_name2 data_type_of_column2
  - - - - - -
  - - - - - -
);
```

The above syntax indicates that column1 is declared as NOT NULL

Example:-

```
CREATE TABLE student
( sid INTEGER
  sname VARCHAR(10)
   age INTEGER NOT NULL
   PRIMARY KEY (sid)
  );
```

Executable Query:

```
INSERT INTO student
VALUES (1, "Ram", 10);
```

Non-executable Query:

```
INSERT INTO student
VALUES (2, "Hari", NULL);
```

ii) UNIQUE constraints:-

The UNIQUE constraint imposes that every value in a column or set of columns be unique. It allows NULL values. It guarantees no two rows of a table can have duplicate values in a specified columns or set of columns.

Syntax:

```
CREATE TABLE <table_name>
(
column_name1 data_type UNIQUE
column_name2 data_type
- - - - - - - - - - -
);
```

The above syntax indicates that column1 is declared as UNIQUE.

Example:-

```
CREATE TABLE student
(
  sid INTEGER,
  sname VARCHAR(20),
  age INTEGER UNIQUE,
  PRIMARY KEY(sid)
);
```

Executable Query:

```
INSERT INTO student VALUES (1, "Rakesh", 20);
INSERT INTO student VALUES (2, "Manoj", 26);
```

Non-executable Query:

```
INSERT INTO student VALUES (3, "Niraj", 25);
```

iii) PRIMARY KEY constraints:-

When an attribute or set of attributes is declared as primary key then the attribute will not accept the null values. Moreover, it will not accept duplicate values. It is to be noted that only one primary key can be defined for each table.

Syntax:

```
CREATE TABLE <table_name>
(
  column_name1 data_type,
  column_name2 data_type,
  ---------- = =
  PRIMARY KEY (column_name1)
);
```

The above syntax indicates that column1 is declared as PRIMARY KEY constraints.

Example:-

```
CREATE TABLE student
(
 sid INTEGER,
 sname VARCHAR (20),
  age INTEGER,
  PRIMARY KEY (sid)
);
```

Executable Query:

```
INSERT INTO student VALUES (1, "Smith", 25);
INSERT INTO student VALUES(2, "Niraj", 27);
```

Non-executable Query

```
INSERT INO student VALUES (NULL, "Opendra", 28);
INSERT INT student VALUES (2, "Mohan", 27);
```

iv) CHECK constraints:-

CHECK constraints allows user to prohibit an operation on a table that would violate the constraint.

Example:

If we need to allow only those students in the table whose age must be an integer range from 20-45.

⇒

```
CREATE TABLE student
(
  sid INTEGER,
  sname VARCHAR(20),
  age INTEGER,
  PRIMARY KEY (sid),
  CHECK (age>=20 AND age<=45)
);
```

Executable query:

```
INSERT INTO student VALUES(1, "sabita", 22),
```

Non-executable query:

```
INSERT INTO student VALUES (2, "Purnima", 50),
```

2. Referential Integrity Constraints:-

The referential integrity constraint in DBMS involves more than one relation.
Suppose we want to ensure that a value appears in one relation for a given set of attribute also appear in a certain set of another relation. This condition is known as referential integrity. To illustrate the concept of referential integrity, let us consider the following relation.

Employee

| Eid | Ename | Salary | Dno. |
|---|---|---|---|
| E01 | Rakesh | 26000 | D1 |
| E02 | Manoj | 29000 | D2 |
| E03 | Niraj | 50000 | D3 |
| E04 | Mahesh | 40000 | D4 |

| Dno | Dname | Location |
|---|---|---|
| D1 | IT | Ghorahi |
| D2 | Finance | Tulsipur |
| D3 | Analyst | Kathmandu |
| D4 | | |

- Defining referential integrity restrict database modification operation such as insert, delete and update.

**Insert:-**

- We can't insert new tuple containing value of foreign key attribute that do not appear in primary key attribute of master table.
- For example, we can't insert new employee that works in D4 department table because the D4 does not exist in department table.

**Delete:-**

- We can't delete tuple containing values of primary key attribute that also appear foreign key attribute of related table.
- For example, we can't delete tuple containing value D2 of Dno from department table because there are employee working in department D2.

**Update:-**

There are two cases of update operation.

**Case1:**

- We can't change the value of primary key attribute if the child table contains related values.
- For example, we can't change value of Dno from $D_1$ to $D_5$ in department table because employee table contains employee working in deparment D1.

Case 2:

- We can't change value of foreign key attribute if the primary key attribute does not contain modified value of foreign key attribute.
- For example, we can't change value of Dno from 03 to 05 in employee table because department table does not contain department 05.

## Referential Integrity in SQL:

Foreign key can be specified as a part of the SQL CREATE TABLE statement by using the ~~foreig~~ FOREIGN KEY clause. By default, a FOREIGN KEY references the PRIMARY KEY attribute of the reference table. SQL also supports REFERENCES clause where a list of attributes of the referenced relation can be specified explicitly.

Example:

```
CREATE TABLE branch
(
  branch_name   VARCHAR(20),
  branch_city   VARCHAR(10),
  assets        INTEGER
  PRIMARY KEY   (branch_name)
);
```

```
CREATE TABLE account
(
    account_number INTEGER,
    branch_name    VARCHAR(10),
    balance        INTEGER,
    PRIMARY KEY    (account_number),
    FOREIGN KEY    (branch_name) REFERENCES branch
);
```

Here, the account table reference the branch table.

## Cascading action in SQL:-

We know that deleting or updating a tuple causes referential integrity to be violated and operations are rejected by DBMS. Cascading action in SQL allows effect to be propagated in related tables rather than rejecting the operations.

Example:

```
CREATE TABLE Book
(
    ISBN    VARCHAR(20);
    title   VARCHAR(20);
    price   INTEGER,
    pid     VARCHAR(5),
    PRIMARY KEY (ISBN),
    FOREIGN KEY (pid) REFERENCES publisher
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
CREATE TABLE Publisher
(
 Pid   VARCHAR (5),
 pname VARCHAR(15),
 City  VARCHAR (20),
 PRIMARY KEY (Pid),
);
```

Due to the ON DELETE CASCADE clause, if a DELETE of a tuple in publisher results in referential integrity constraint violation then the delete is propagated to the Book relation, deleting the tuple that refers to the publisher that was deleted. Similarly update operation is performed.

which database has to satisfy always.
is a condition, jisse database ko consistency kai preserve garos.

3. Assertions :-

An assertion is a predicate expressing a condition that we wish the database always to satisfy. Domain constraints and referential constraints are special form of assertion. We have paid attention to form assertions because they must be easily tested and apply to a wide range of database applications.

Syntax:

```
CREATE ASSERTION <assertion_name>
        CHECK <predicate>
```

Example:

The publisher id of book relation is always not null, since each book must be published by any publishers.

⇒

```
CREATE ASSERTION NoPub CHECK
    (NO EXISTS
        (SELECT *FROM Book
         WHERE Pid IS NULL));
```

Above assertion ensures that there is no book that does not have Publisher. Lets take instance of Book relation.

| ISBN | title | price | pid |
|---|---|---|---|
| 001 | C++ | 400 | 01 |
| 002 | Java | 450 | 02 |
| 003 | DBMS | 500 | NULL |

In the above Book relation the pid of DBMS is null due to which assertion is violated and we cannot modify database further. Assertion can be dropped using DROP ASSERTION command. For eg: DROP ASSERTION N

statement which is executed when some event is occurred.

4. Triggers :-

A trigger is a statement that the system executes automatically as a side effect of a modification to the database. A database that has a set of associated trigger is called an active database. To design a trigger, following are the three requirements:

i) Event:

A change to the database that activates the trigger.

ii) Condition:

Trigger performs some actions only if a specified condition matches at the occurance of the event.

iii) Action:

A procedure that executed when the trigger is activated and its condition is true.

Syntax:-

```
CREATE TRIGGER <trigger_name>
    <time event>
    ON <table list>
    WHEN <predicate>
        <action_name>
```

Three event types:
i) Insert
ii) Delete
iii) Update

Two triggering times:
i) Before the event
ii) After the event

Two granularities:
i) Execute for each row.
ii) Execute for each statements

Example:
For every prepaid account whose balance is less than or equal to zero, then account is automatically blocked.

```
CREATE TRIGGER overdraft
AFTER UPDATE ON account
REFERENCING NEW ROW as nrow
FOR EACH ROW
WHEN nrow.balance <= 0
UPDATE account SET blocked = 'T'
```

# Security and Authorization

## Security Violation:-

Database security refers to protection from malicious access. Following are the form of malicious access:

a) Unauthorized reading of data
b) Unauthorized modification of data
c) Unauthorized destruction of data

To protect a database, we can take security measures at following levels.

a) Database system
b) Operating system
c) Network
d) Physical
e) Human

## Authorization:-

- Authorization is the function of specifying access right to resources.
- We may assign a user several forms of authorization on parts of the database.

such as:

i) Read authorization allows reading, but not modification of data.
ii) Insert authorization allows insertion of new data but

not modification of existing data.

iii) Update authorization allows modification, but not deletion of data.

iv) Delete authorization allows deletion of data.

- We may assign the user all, non, or a combination of these types of authorization.
- In addition to these forms of authorization for access to data, we may grant a user authorization to modify the database schema.
  - i) Index authorization allows the creation and deletion of indices.
  - ii) Resource authorization allows the creation of new relations.
  - iii) Alteration authorization allows the addition or deletion of attributes in a relation.
  - iv) Drop authorization allows the deletion of relations.

## Encryption and Authorization:-

- The various schemas that the database system may use for authorization may still not provide sufficient protection for highly sensitive data.
- In such a case, data may be stored in encrypted form.
- It is not possible for encrypted data to be read unless the reader knows how to decrypt them.

- Authentication refers to the task of varying the identity of a person / software connecting to a database.
- The simplest form of authentication consists of a secret password which must be presented when a connection is opened to a database.

# Theory of Database Design (Relational Database Design)

## Functional Dependency:-

Functional dependencies are the relationship among the attribute within the relation. Functional dependencies provide a formal mechanism to express constraints between attributes.

## Formal Definition:-

- Let A and B are attributes of a relation R. If each value of B is associated with exactly one value of A then B is said to be functionally dependent on A.
- It is denoted by $A \rightarrow B$.
- So the functional dependency $A \rightarrow B$ holds if whenever two tuples have the same value for A they must have same value for B.
- More formally, for any two tuples $t_1$ and $t_2$ in any relation instance $r(R)$.

if $t_1[A] = t_2[A]$
then $t_1[B] = t_2[B]$

## Example:

Student (sid, sname, address, age)

Here, sname, address and age are functionally dependent on sid. Meaning is that each student sid uniquely determines the value of attribute, name, address & age.

- This can be expressed as:
  sid → sname
  sid → address
  sid → age

Example:
Let us consider the instance of following table.

| A | B | C | D | E |
|---|---|---|---|---|
| a | 2 | 3 | 4 | 5 |
| 2 | a | 3 | 4 | 5 |
| a | 2 | 3 | 6 | 5 |
| a | 2 | 3 | 6 | 6 |

Is the given functional dependencies valid?
A → BC (valid)
DE → C (valid)
C → DE (invalid)
BC → A (valid)

## Types of functional dependencies:-

There are many types of functional dependencies; depending on several criteria:

i) Fully functional dependency
ii) Partial functional dependency
iii) Trivial and non-trivial dependency
iv) Transitive dependency

i) Fully functional dependency:-

In a relation R, an attribute Y is said to be fully functionally dependent on attribute X if it is functionally dependent on X and not functionally dependent on any proper subset of X.

Example:

- In a relation student (sid, major, fee), fee is not fully functionally dependent on concatenated key (sid, major) because fee is functionally dependent on the major.

ii) Partial functional dependency:-

For a relation R, in a functional dependency $X \rightarrow Y$, Y is said to be partial functional dependent on X if by removal of some attributes from X, the dependency still holds.

Example:

The dependency [emp_id, project_no] → emp_name is partial because emp_id → emp_name also holds.

iii) Trivial and Non-trivial dependency :-

- A functional dependency $X \rightarrow Y$ is said to be trivial dependency if Y is subset of X (not necessarily a proper subset).
- The functional dependency that satisfied by all the relation is called trivial dependency.

Example:

$AB \rightarrow A$ is satisfied by all relations involving attribute A.

iv)

- The functional dependency that is not trivial is said to be non-trivial dependency.

iv) Transitive dependency:-

A transitive dependency exists when there is an intermediate functional dependency.

- If A functionally determines B, and B functionally determines C then A functionally determines C.
  i.e. $A \rightarrow B$, $B \rightarrow C$ then $A \rightarrow C$

Example:

Consider the following relation.

supplier (sname, item, price, gift item)

- In this relation the sname and item can determine the price and price obviously determines gift item.
- Then the functional dependency can be represesented as follows:

sname, item $\rightarrow$ price

price $\rightarrow$ gift item.

## Functional Dependency Inference Rules (Armstrong's Axioms)

1) Reflexivity Rule:
If $\alpha$ is a set of attributes and $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ holds.

2) Augmentation Rule:
If $\alpha \rightarrow \beta$ holds and $\gamma$ is a set of attributes then $\gamma\alpha \rightarrow \beta$.

3) Transitivity Rule:
If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds then $\alpha \rightarrow \gamma$ holds.

4) Union Rule:
If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.

5) Decomposition Rule:
If $\alpha \rightarrow \beta\gamma$ holds then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.

6) Pseudo transitivity Rule:
If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds then $\alpha\gamma \rightarrow \delta$ holds.

## Closure of attribute sets:-

- Let $\alpha$ be a set of attributes.
- We call the set of attributes functionally determined by $\alpha$ under a set $F$ of functional dependencies, the

closure of $\alpha$ under F.
We denote it by $\alpha^+$.
The algorithm to compute $\alpha^+$ is as follows:

```
result = α
repeat
    for each functional dependency β→γ in F do
    begin
        if β ⊆ result then
        result = result ∪ γ
    end
until (result doesn't change).
```

Example:
Given a relational schema R(A,B,C,G,H,I) and the set of functional dependencies.

$A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H$

then

$(AG)^+ = ?$

Sol:

result = AG
$A \rightarrow B$ causes the result = ABG
$A \rightarrow C$ causes the result = ABCG
$CG \rightarrow H$ causes the result = ABCGH

$CG \rightarrow I$ causes the result = ABCGHI
$B \rightarrow H$ causes the result = ABCGHI

## Equivalence of functional dependencies (Cover):-

- If F and G represents two sets of functional dependencies defined over the same relational schema, then F and G are equivalent if $F^+ = G^+$.
- Whenever $F^+ = G^+$, then F covers G and vice-versa.

Example:

Lets take two sets of functional dependencies:

$F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$

and, $G = \{A \rightarrow CD, E \rightarrow AH\}$

Test whether these dependencies are equivalent or not.

Sol:

To test G covered by F.

F: $A \rightarrow C$
$AC \rightarrow D$
$E \rightarrow AD$
$E \rightarrow H$

G: $A \rightarrow CD$
$E \rightarrow AH$

$A \rightarrow ACD$
$AC \rightarrow ACD$
$E \rightarrow ACDEH$
$E \rightarrow ACDEH$

$A \rightarrow ACD$
$E \rightarrow ACDEH$

Thus, these two functional dependencies are equivalent.

## Non-redundant cover:

Consider two sets of functional dependencies F and G defined over the same relational schema, if G covers F and no proper subset H of G is such that $H^+ = G^+$, then G is a non-redundant cover of F.

## Minimal Cover (Non-reducable Cover):

A set of non-redundant functional dependencies which is obtained by removing all redundant functional dependencies using the functional dependencies inference rule is termed as minimal cover.

Example: Let us consider the following dependencies.

$R(W, X, Y, Z)$

$X \rightarrow W$

$WZ \rightarrow XY$

$Y \rightarrow WXZ$

Find the minimal cover of above dependencies.

Soln:

At first decompose the given dependencies.

$X \rightarrow W$

$WZ \rightarrow X$

$WZ \rightarrow Y$

$Y \rightarrow W$

$Y \rightarrow X$

$Y \rightarrow Z$

Step 1: Find $X^+ = XW$

Find $X^+$ by ignoring $X \rightarrow W$, then $X^+ = X$

and soon.

After removing redundant functional dependencies

$X \rightarrow W$

$WZ \rightarrow Y$

$Y \rightarrow X$

$Y \rightarrow Z$.

Again checking the left side of the functional dependency.

$(WZ)^+ = WXYZ$

Now,

$W^+ = W$

$Z^+ = Z$

Thus, the minimal cover of given fd is:

$X \rightarrow W$

$WZ \rightarrow Y$

$Y \rightarrow XZ$.

## Normalization:-

The process of decomposing bad relation by breaking up their attributes into smaller relations is called normalization.

## Purpose of Normalization:-

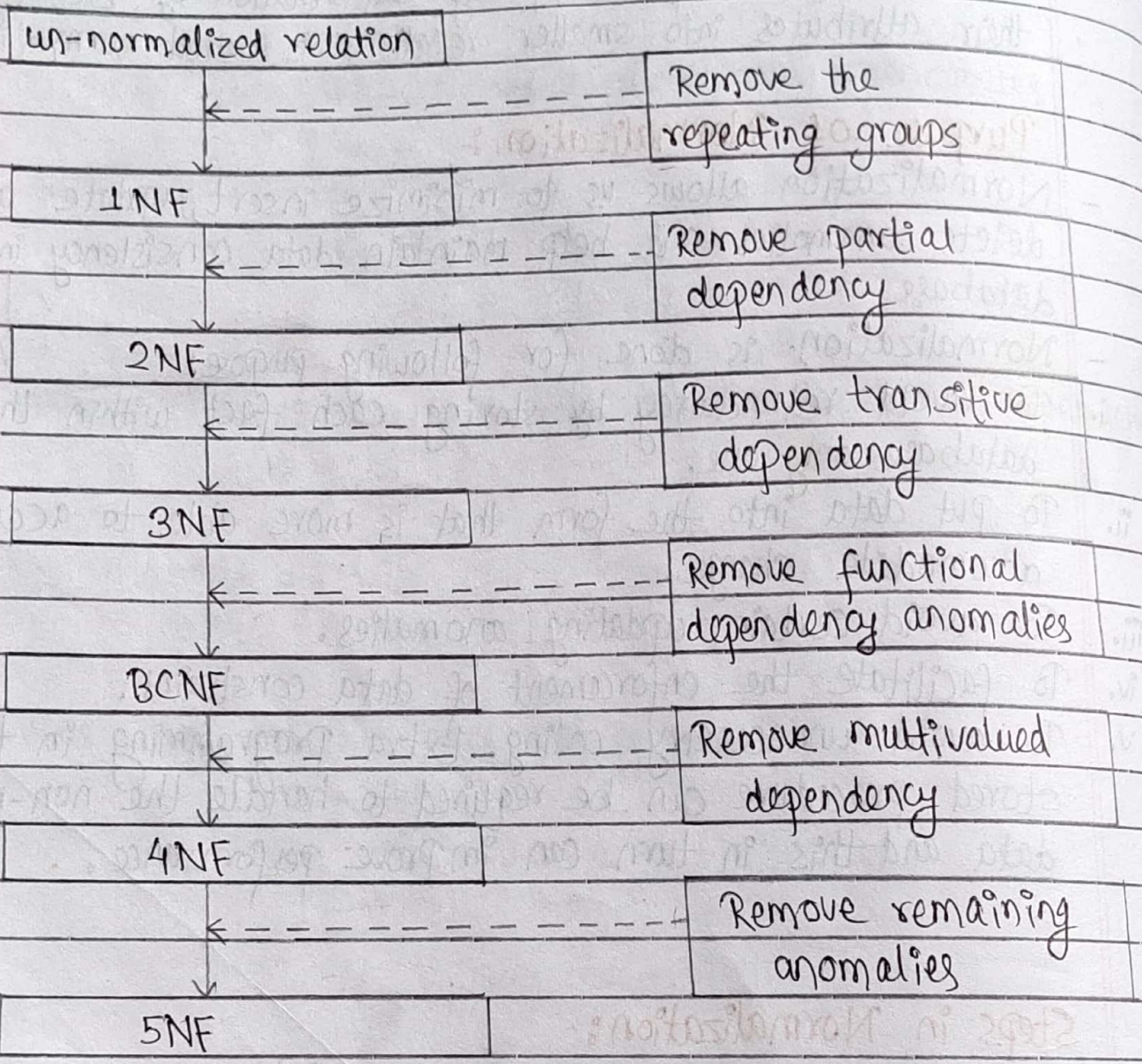
- Normalization allows us to minimize insert, update and delete anomolies and help maintain data consistency in the database.
- Normalization is done for following purpose:

i. To avoid redundancy by storing each fact within the database only one.
ii. To put data into the form that is more able to accurately accomodate charge.
iii. To avoid certain updating anomalies.
iv. To facilitate the enforcement of data constraint.
v. To avoid unnecessary coding. Extra programming in trigger, stored procedure can be required to handle the non-normalized data and this in turn can improve performance.

## Steps in Normalization:-

- The degree of normalization is defined by normal forms.
- The normal forms in an increasing level of normalization are 1NF, 2NF, 3NF, 4NF and 5NF.
- Each normal form is a set of conditions on a schema that guarantees certain properties relating to redundancy and update anomalies.

- In general, 3NF is good enough.




[Fig:- Steps in normalization]

## 1NF (First Normal Form):-

A relation is said to be in first normal form if and only if all domains of the relation contains only atomic values. It does not allow multi-valued attributes.

Example:

Let us consider a relation

Student

| sid | name | courses |
|---|---|---|
| 101 | Ram | DBMS, TOC, CS |
| 102 | Hari | OS, SAD, RT |

Above relation is not in 1NF. The normal table for above table is:

| sid | name | courses |
|---|---|---|
| 101 | Ram | DBMS |
| 101 | Ram | TOC |
| 101 | Ram | CS |
| 102 | Hari | OS |
| 102 | Hari | SAD |
| 102 | Hari | RT |

## 2NF (Second Normal Form):-

A relation is said to be in second normal form if and only if:

a) it is already in 1NF

b) every non-prime attribute is fully dependent on the prime (primary) attribute. i.e. there is no partial dependency.

Example:

Given a relation R(A,B,C,D) and for

$AB \rightarrow D$

$B \rightarrow C$

then the above relation is not in 2NF. Since C is partially dependent on the prime attribute B. The relation after converting 2N is:

$R_1$

| A | B | D |
|---|---|---|
| | | |
| | | |

B

| B | C |
|---|---|
| | |
| | |

Example2:

Courses (course_no, title, loc, time)

course_no → title

course_no, time → loc

Here,

candidate key = {course_no, time}

non-prime attribute = {loc, time}

The relation after converting 2NF is:

| course_no | time | loc |
|---|---|---|
| | | |
| | | |

| course_no | title |
|---|---|
| | |
| | |

## 3NF (Third Normal Form):-

A relation is said to be in third normal form if and only if:

a) it is already in 2NF

b) every non-prime attribute is non-transitively dependent on the prime attribute.

Example:

Let us consider the following relation: R(A,B,C,D) and following functional dependency.

$AB \rightarrow C$

$C \rightarrow D$

The relation is not in 3NF since there is transitive dependency. The normalized relation after removing transitive dependency.

$R_1$

| A | B | C |
|---|---|---|
| | | |
| | | |

$R_2$

| C | D |
|---|---|
| | |
| | |

Example 2:

Let us consider a relation student

| Sid | Sname | Age | sex | hostel_name |
|---|---|---|---|---|
| | | | | |
| | | | | |

and fd.

sid → sname, age, sex, hostel_name

sex → hostel_name

| sid | sname | age | sex |
|---|---|---|---|
| | | | |
| | | | |

| sex | hostel_name |
|---|---|
| | |
| | |

## Boyce Code Normal Form (BCNF):-

A relation R is said to be in Boyce Code Normal Form with respect to all the functional dependencies of the form $\alpha \rightarrow \beta$ where $\alpha \subseteq R$ and $\beta \subseteq R$ if at least one of following condition holds:

a) $\alpha \rightarrow \beta$ is a trivial functional dependency (i.e. $\beta \subseteq \alpha$)

b) $\alpha$ is candidate key for schema R.

Simply, a relation R is in BCNF if and only if every determinant (left side) is a candidate key.

Example:

Let us consider a relation R(A,B,C) with the following fd.

$AB \rightarrow C$

$C \rightarrow B$

Here, candidate key = AB

non-prime attribute = C

In above relation, the 3NF doesn't fail because there is no transitive dependency. There is a anamolies when non-prime attributes determines prime attributes. Thus, BCNF doesn't hold. The relation after converting in BCNF is:

$R_1$

| A | C |
|---|---|
| | |
| | |

$R_2$

| C | B |
|---|---|
| | |
| | |

Example:

Bank (br_code, name, agent)

agent → br_code

name, br_code → agent

Here,

candidate key = {name, br_code}

non-prime attribute = {agent}

Here the agent is determinant but it alone cannot determine all the other attributes. Thus, the table in BCNF is:

| agent | br_code |
|---|---|
| | |
| | |

| name | agent |
|---|---|
| | |
| | |

## BCNF and 3NF:

- All BCNF are in 3NF but not all 3NF are in BCNF.
- BCNF doesn't make any reference to the concept of full or partial dependency.
- BCNF is stronger form of normalization than 3NF which allows right side of the fd to be the prime attribute.
- Thus, every left side of fd in a table must be a super key.

## Multi-valued f.dependency:-

Multivalued dependency do not rule out the existence of certain tuple. Instead, they require that other tuples of a certain form present in the relation. For this reason, multivalued dependency are referred to as tuple generating dependencies.

- Formally, let r(R) be a relation schema and $\alpha \subseteq R$ and $\beta \subseteq R$
- Then multivalued dependency $\alpha \twoheadrightarrow \beta$ hold on R if in any legal instance of relation r(R) for all pairs of tuples $t_1$ and $t_2$ in r, such that $t_1[\alpha] = t_2[\alpha]$, there exists tuple $t_3$ & $t_4$ in r such that

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R-\beta] = t_2[R-\beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R-\beta] = t_1[R-\beta]$$

Example:

Let us consider a relation $R(\alpha, \beta, \gamma)$ then if $\alpha \rightarrow\rightarrow \beta$ then following condition should satisfy.

| $\alpha$ | $\beta$ | $\gamma$ |
|---|---|---|
| a | $b_1$ | $c_1$ |
| a | $b_2$ | $c_2$ |
| a | $b_1$ | $c_2$ |
| a | $b_2$ | $c_1$ |

Example:

Let us consider the following relation

Apply (citizen_no, college_name, hobby)

The multivalued fd exist.

citizen_no $\rightarrow\rightarrow$ college_name

If following condition satisfies:

| citizen_no | college_name | hobby |
|---|---|---|
| 101 | AMC | Football |
| 101 | KMC | TT |
| 101 | AMC | TT |
| 101 | KMC | Football |

## Trivial Multivalued Dependency:-

If the multivalued dependency $\alpha \rightarrow\rightarrow \beta$ is satisfied by all attributes on schema R then $\alpha \rightarrow\rightarrow \beta$ is trivial multivalued dependency on R.

i.e. if $\alpha \rightarrow\rightarrow \beta$ is trivial then $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$.

## 4NF (Fourth Normal Form):-

The goal of 4NF is to eliminate non-trivial multivalued dependency from the table by projecting them onto separate smaller tables. This eliminates the ~~operating~~ updating anamolies associated with the multivalued dependencies.

A table R is in 4NF if and only if it satisfies following two conditions simultaneously:

a) R is already in BCNF &

b) if it contains no multivalued dependency.

Example:

Consider the following table.

| citizen_no | college_name | hobby |
|---|---|---|
| 101 | AMC | Football |
| 101 | KMC | TT |
| 101 | AMC | TT |
| 101 | KMC | Football |

This table is not in 4NF because there exists multivalued dependency between (citizen_no and college_name) & (citizen_no and hobby)

i.e. citizen_no →→ college_name
citizen_no →→ hobby

Thus, after splitting the table, we get the table in 4NF as follows:

| citizen_no | college_name |
|---|---|
| 101 | AMC |
| 101 | KMC |

| citizen_no | hobby |
|---|---|
| 101 | Football |
| 101 | TT. |

## Properties of decomposition:

There are two essential properties of decomposition
a) Lossless decomposition preservation
b) Dependency preservation

### a) Lossless decomposition:-

- Let r(R) be a relation schema and let F be a set of functional dependency on r(R).
- Let $R_1$ and $R_2$ form a decomposition of R.
- We say that the decomposition is a lossless decomposition if there is no loss of information by replacing r(R) with two relation schemas $r_1(R_1)$ and $r_2(R_2)$.
- In relational algebra it can be stated as follows:

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

- If a decomposition is not a lossless then it is called lossy decomposition.

Example:

Decomposition of a relation R = (A, B, C) into two relations $R_1$ = (A, B) and $R_2$ = (B, C) as,

| A | B | C |
|---|---|---|
| 1 | p | q |
| 2 | q | b |

r

$\Rightarrow$

| A | B |
|---|---|
| 1 | p |
| 2 | q |

$\Pi_{A,B}(r)$

| B | C |
|---|---|
| p | q |
| q | b |

$\Pi_{B,C}(r)$

$\Rightarrow$

| A | B | C |
|---|---|---|
| 1 | p | q |
| 2 | q | b |

$\Pi_A(r) \bowtie \Pi_B(r)$

b) Dependency preservation:- dependency closure same hunpryg

- Let F be a set of functional dependencies or a schema R and let $R_1, R_2, \ldots, R_n$ be a decomposition of R.
- Let $F_1, F_2, \ldots, F_n$ are functional dependencies on $R_1, R_2, \ldots, R_n$ respectively and let $F' = F_1 \cup F_2 \cup \ldots \cup F_n$.
- F' is a set of functional dependencies on schema R but in general $F' = F$.
- However if $F' \neq F$, it may be that $F'^+ = F^+$.
- If the later condition is true then this decomposition is said to be dependency preserving decomposition.

5NF (Fifth Normal Form):-

A relation R is in fifth normal form (5NF) if and only if the following conditions are satisfied simultaneously.

a) R is already in 4NF.

b) It cannot be further non-loss decomposed.

UNIT-3

# Transaction Processing

## Transaction:-

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- In other words, collection of operation that form a single logical unit of work are called transactions.
- A transaction is delimited by statements (or function calls) of the form 'begin' transaction and 'end' transaction.
- It consists of all operations executed between the begin transaction and end transaction.
- A transaction access data using two operations:
  a) read
  b) write

Example:

Let $T_i$ be a transaction that transfer $50 from account A to account B. This transaction can be defined as:

```
T1 : read(A);
     A:=A-50;
     write(A);
     read(B);
     B:=B+50;
     write(B);
```

## Read / Write operations of transaction:

Read and write are two basic operations of a transactions:

1. Read (X):-
   - This transfer the data item X from the database to a local buffer belonging to the transaction that executed the read operation.
   - For this it performs the following sequence of operations:
     i, Find the address of the disk block that contains item X.
     ii, Copy that disk block into a buffer in main memory (if that disk block is already in some main memory buffer).
     iii, Copy item X from the buffer to the program variable named X.

2. Write (X):-
   - This transfer the data item X from the local buffer belonging to the transaction that executed the write back the database.
   - For this, it performs following sequence of operations:
     i, Find the address of the disk block that contains item X.
     ii, Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).

iii, Copy item X from the program variable named X into its correct location in the buffer.
iv, Stores the upload block from the buffer back to disk.

## Desirable Properties of a Transaction (ACID properties):-

- To ensure the integrity of the data, the database system must maintain some desirable properties of the transaction.
- These properties are known as ACID properties.
- The ACID properties are explained as follows:

### i. Automicity :-

- Either all operations of a transaction are rejected properly in the database or non are.
- The basic idea behind ensuring automicity is as follows:
- The database keeps track of the old values of any database on which a transaction performs a write and if the transaction does not complete its execution, the old values are restored to make it appear as though the transaction never executed.
- It is the responsibility of the transaction recovery sub-system of a DBMS to ensure automicity.

### ii. Consistency:-

- The consistency property ensures that any transaction will bring the database from one valid state to another.
- Execution of a transaction in isolation preserves the consistency of the database.

- The consistency property of the transaction is maintained by application programmer who develops the program for database.

iii. **Isolation :-** kunai 2ta transaction independently ek arka lai interfere nagarera access huna paunu parxa.

- Even though multiple transaction may execute concurrently the system guarantees that for every pair of transaction $T_i$ and $T_j$, it appears to $T_i$ that either $T_j$ finished execution before $T_i$ started or $T_j$ started execution after $T_i$ finished.
- Thus each transaction is unaware of other transactions executing concurrently in the system.
- The isolation property is maintained by concurrency control component.

2ta transaction haru sangai independently run hunxa ra one should not affect other.

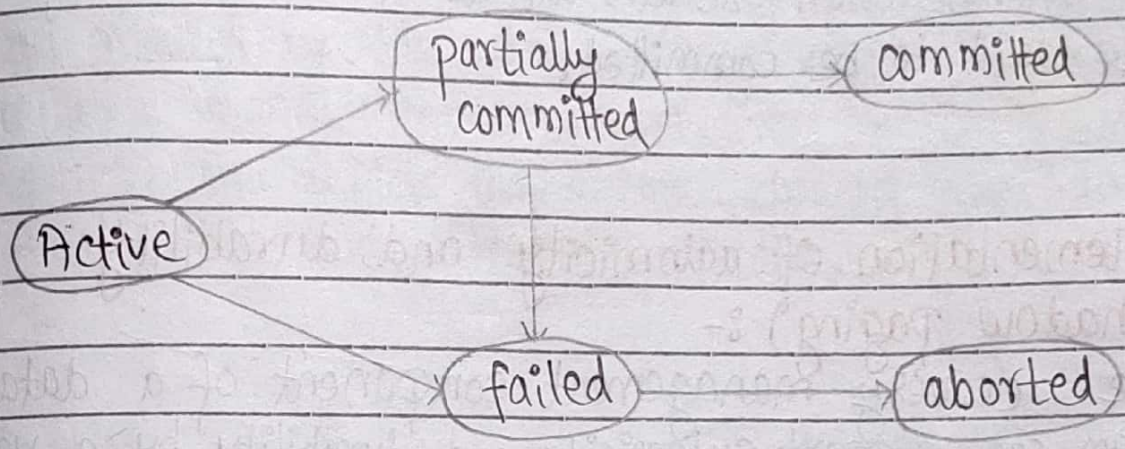
iv. **Durability :-** valid transaction le gareko action long time samma durable huna parxa.

- After a transaction completes successfully, the changes it has made to the database persist, even if there are system failure.
- The durability property guarantees that, once a transaction complete successfully, all the updates that it carried out on the database presists even if there is system failure after the transaction completes execution.
- It is the responsibility of recovery management component to ensure durability.

valid transaction le change gareko kura permanent hunxa.

## Transaction state:-

- Whenever a transaction is submitted to a DBMS for execution either it executes successfully or fails due to some reasons.
- During its execution, a transaction passes through various states that are active, partially committed, committed, failed and aborted.
- Figure below shows the state transaction diagram that describes how a transaction passes through various states during its execution.




[Fig:- States of a transaction]

### i. Active state:-

- It is the initial state of the transaction.
- In this state the transaction is being executed.

### ii. Partially committed :-

- When a transaction executes its final operation, it is said to be in this state.

iii. Failed :-
- A transaction goes in failed state after the discovery that normal execution can no longer proceed.

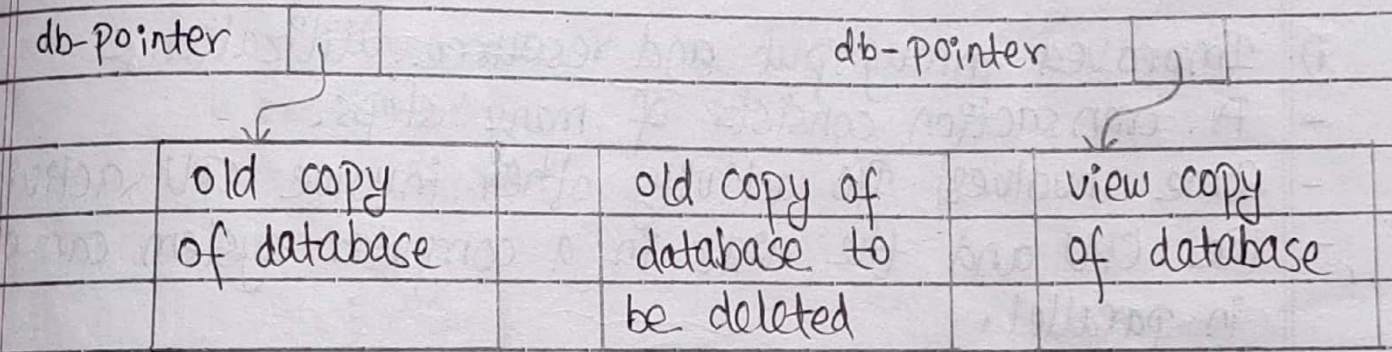
iv. Aborted :-
- After the transaction has been rolled back and the database has been restored to its prior to the start of the transaction.

v. Committed :-
- If a transaction executes all its operation successfully, it is said to be committed.

## Implementation of automicity and durability (shadow paging) :-

- The recovery management component of a database system can support automicity and durability by a variety of schemas.
- Here we consider a simple but extremely efficient scheme called the shadow copy scheme.
- The shadow copy scheme is based on making the copies of the database called shadow copies.
- It assumes that only one transaction is active at a time.
- It also assumes that the database is simply a file on disk.
- A pointer called db-pointer maintained on disk which points to the current copy of the database.
- In the shadow-copy scheme, a transaction that wants to

update the database first creates a complete copy of the database.
- All updates are done on the new database copy leaving the original copy, the shadow copy, untouched.
- If in any point the transaction has aborted, the system deletes the new copy.
- The old copy of the database has not affected.
- If the transaction completes then the old copy of the database is then deleted.
- Thus, the automicity and durability property of transaction are ensured by the shadow-copy implementation of the recovery management component.
- Figure below depicts the scheme, showing the 'database state before and after the update.




a) before update b) after update

[Fig: Shadow-copy technique for automicity and durability]

## Concurrent Execution of Transaction:-

- Concurrently means that task A and task B both need to happen independently of each other and A starts running and the B starts before A is finished.
- Transaction processing system usually allow multiple transaction to run concurrently.
- There are different ways of accomplishing concurrency.
  - i) Parallel processing more than one CPU
  - ii) Interleaved processing or task switching. one CPU
- Allowing multiple transaction to update data concurrently causes several complications with consistency of the data.
- However, there are two good reasons for allowing concurrency.

i) Improved throughput and resource utilization:-

- A transaction consists of many steps.
- Some involves I/O activity other involve CPU activity.
- The CPU and the disks in a computer system can operate in parallel.
- Therefore I/O activity can be done in parallel with processing at CPU.
- This increases the throughput and utilization of processor and I/O.

ii) Reduced waiting time :-
Concurrent execution reduces the unpredictable delay in running transactions.

## Transaction schedule:

Transaction schedule is a sequence that includes the chronological order in which instructions of concurrent transactions executed. A schedule for a set of transaction must consist of all instructions of those transactions. A schedule mush preserve the order in which the instructions appeared in each individual transactions.

Example:

Consider the simplified banking system. Transaction $T_1$ transfers $50 from account A to account B. It is defined as:

```
T1 = read (A);
     A := A-50;
     write (A);
     read (B);
     B := B+50;
     write (B);
```

Transaction $T_2$ transfers 100% of the balance from account A to account B. It is defined as:

```
T2 : read (A);
     temp := A * 0.1;
```

```
A := A - temp;
write (A);
read (B);
B := B + temp;
write (B);
```

Then the simplified schedule for these two transaction is:

| $T_1$ | $T_2$ |
|---|---|
| read (A) | |
| A := A-50 | |
| write (A) | |
| read (B) | |
| B := B+50 | |
| write (B) | |
| | read (A) |
| | temp := A * 0.1 |
| | A := A - temp |
| | write (A) |
| | read (B) |
| | B := B + temp |
| | write (B) |

In multi-user environment, multiple transactions can be executed concurrently. On the basis of whether operations from different transaction interleaved or not, schedule can be divided into two categories:

i) Serial schedule
ii) Non-serial schedule

1) Serial schedule :-

A schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle then next transaction is executed. Transactions are ordered in one after another. This type of schedule is called serial schedule because transactions are executed in a serial manner. The serial schedule is not good for multiuser environment.

Example:

Let us schedule the above banking system's transactions $T_1$ and $T_2$ as:

| $T_1$ | $T_2$ |
|---|---|
| read (A) | |
| A := A-50 | |
| write (A) | |
| read (B) | |
| B := B+50 | |
| write (B) | |
| | read (A) |
| | temp := A*0.1 |
| | A := A-temp |
| | write (A) |
| | read (B) |
| | B := B+temp |
| | write (B) |

4) Non-serial schedule :-

A schedule where the operations from a set of concurrent transactions are interleaved is called non-serial schedule. Although non-serial schedule are important to achieve efficient throughput in multiuser environment, they do not produce correct result always. If two transactions are independent then execution of transaction in any order do not effect the result. If the two transactions are mutually dependent i.e. working in the same data, result may vary.

Example:

The non-serial schedule for above banking system may occur in the form:

| $T_1$ | $T_2$ |
|---|---|
| Read (A) | |
| A := A-50 | |
| write(A) | |
| | Read(A) |
| | temp := A * 0.1 |
| | A := A - temp |
| | write (A) |
| Read (B) | |
| B := B+50 | |
| write (B) | |
| | Read (B) |
| | B := B+temp |
| | write (B) |

[Fig: Non-serial schedule]

i) diff^ transaction
ii) same data ma kaam garnu parcha
iii) write operation



## Conflict Schedule:-

Concept of conflict schedule is based on conflict operations. Two operations in a schedule are ~~said~~ set to be conflict if they satisfy all three of the following conditions:

i) They belong to different transactions.
ii) They access the same data item say 'x'
iii) At least one of the operation is write operation.

A schedule is called conflict schedule if it contains at least one pair of conflicting operations on it.

Example:

$S_1$ : $R_1(X)$, $W_2(X)$, $W_3(X)$ conflict schedule.
$S_2$ : $R_1(X)$, $R_2(X)$, $R_3(X)$ non-conflict schedule
$S_3$ : $R_1(X)$, $W_2(Y)$, $R_3(X)$ non-conflict schedule.

same initial value, $T_i$ to $T_j$ = $T_i$ to $T_j$ write last me xa vane arkoma pani lekha write

## Equivalent Schedule:-

There are several way to define equivalence of schedule. The simplest definition of equivalent schedule involves comparing the effect of the schedule on the database. These categories of equivalent schedule are:

### i) Result Equivalent schedule:-

Two schedules are called result equivalent if they produce the same final state of the database. However, two different schedules may accidently produce the same final state. So, result equivalent alone cannot be used to define equivalence of schedule.

ii) View Equivalence schedule:-

Two schedules $S_1$ and $S_2$ are said to be view equivalent if transactions in both schedules performs similar actions in similar manner. Two schedules will be equivalent if they meet following three conditions:

a) For every data item X, if $T_i$ reads initial value of X in $S_1$ then $T_i$ must also read initial value of X in $S_2$.

b) For every data item X, if $T_i$ reads value written by $T_j$ in $S_1$ then $T_i$ must also read value written by $T_j$ in $S_2$.

c) For every data item X, if $T_i$ performs final write on data value in $S_1$ then $T_i$ must also perform final write on data value in $S_2$.

Example:

$S_1$

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | |
| write(A) | |
| | read(A) |
| | write(A) |
| read(B) | |
| write(B | |
| | read(B) |
| | write(B) |

$S_2$

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | |
| write(A) | |
| Read(B) | |
| write(B) | |
| | Read(A) |
| | write(A) |
| | Read(B) |
| | write(B) |

[Fig:- View equivalent schedule. $S_1$ & $S_2$]

iii) Conflict Equivalent Schedule :-

Two schedules having more than one transactions with conflicting operations are said to be conflict equivalent if and only if both schedules contains same set of transactions and the order of pairs of operations is maintained in both schedules.

Example:

$S_1$ : $R_1(X)$ $R_2(X)$ $W_2(X)$ $W_1(X)$ $R_3(Y)$

$S_2$ : $R_1(X)$ $R_2(X)$ $R_3(Y)$ $W_2(X)$ $W_1(X)$

$S_3$ : $R_1(X)$ $R_2(X)$ $W_1(X)$ $R_2(Y)$ $W_2(X)$

Conflicting pairs of above schedules are :

$S_1 = \{R_1(X), W_2(X)\}$ $\{R_2(X), W_1(X)\}$

$S_2 = \{R_1(X), W_2(X)\}$ $\{R_2(X), W_1(X)\}$

$S_3 = \{R_2(X), W_1(X)\}$ $\{R_1(X), W_2(X)\}$.

Here, schedule $S_1$ and $S_2$ are conflict equivalent because order of all conflicting operations are same in both schedules.

V.V. Imp (Imp)

## Serializibility :-

Serializibility is the process of checking whether the given non-serial schedule is serializable or not. A given non-serial schedule of n transactions is serializable if it is equivalent to some serial schedule i.e. if a non-serial schedule produce the same result as of the serial schedule then the given non-serial schedule is said to be serializable.

A schedule that is not serializable is called non-serializable schedule.

conflict pair serial

## Conflict serializability :-

A non-serial schedule is said to be conflict serializable when the schedules is conflict equivalent to one or more serial schedules. In another words, a schedule is conflict serializable if and only if its precedence graph or serializibility graph is acyclic, for only committed transaction is considered. If a non-serial schedule S can be transformed into a serial schedule S' by a series of swap of non-conflicting instructions then we say that schedule S is conflict serializable.

Let us consider a schedule S, if instruction I and J refer to different data items then we can swap I and J without affecting the result of any instruction in the schedule.

However, if I and J refer to the same data item X, then the order of the two instruction may matter. Since, we are dealing with only read and write instructions, there are four cases that we need to consider.

1. I = read(X), J = read(X):-
The order of I and J does not matter.

2. I = read(X), J = write(X):-
If I comes before J, then transaction $T_i$ reads the value of X that before updating value by transaction $T_j$. If J comes before I, then transaction $T_i$ reads the value of X that is updated by transaction $T_j$. Thus, order of operations of I and J matters.

3. I = write(X), J = read(X):-
The order of operations of I and J matters similar to those of the case 2.

4. I = write(X), J = write(X):-
Since both instructions are write operations, the order of the instructions doesnot matter because the result of only the later of the two write transactions is preserved in the database.

Example:
Consider the following two schedules,

$S_1$

| $T_1$ | $T_2$ |
|---|---|
| R(X) | |
| W(Y) | |
| commit | |
| | R(X) |
| | W(X) |
| | commit |

$S_2$

| $T_1$ | $T_2$ |
|---|---|
| R(X) | |
| | R(X) |
| W(Y) | |
| commit | |
| | W(X) |
| | commit |

Schedule $S_2$ is conflict serializable because there exists a serial schedule $S_1$ that is conflict equivalent with $S_2$.

## Testing for conflict serializibility of a schedule :-

Simplest and efficient way to determine conflict serializibility of a schedule is precedence graph. A precedence graph consists of a pair $G = (V, E)$ where, V is a set of vertices and E is a set of edges. The set of vertices consists of all the transactions participating in the schedule. The set of edges consists of all edges of the form $T_i \rightarrow T_j$ for which one of three condition holds:

i) $T_i$ executes write(X) before $T_j$ executes read(X).
ii) $T_i$ executes read(X) before $T_j$ executes write(X).
iii) $T_i$ executes write(X) before $T_j$ executes write(X).

There is a simple algorithm for determining whether a particular schedule is conflict serializable or not. If precedence graph is acyclic then serializibility order can be obtained by a topological sorting of the graph.

Algorithm:

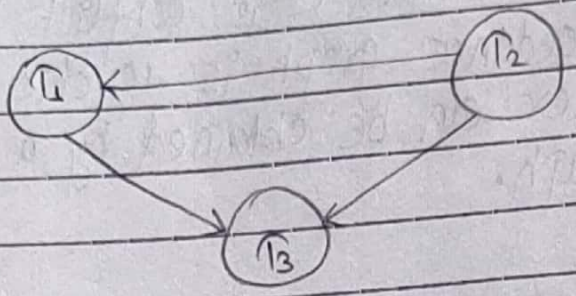
1. For each transaction $T_i$ participating in schedule S, create a node labelled $T_i$ in the precedence graph.
2. For each case in S where $T_j$ executes a read(X) operation after $T_i$ executes a write(X) operation, create an edge ($T_i \rightarrow T_j$) in the precedence graph.
3. For each case in S where $T_j$ executes a write(X) operation after $T_i$ executes a read(X) operation, create an edge ($T_i \rightarrow T_j$) in the precedence graph.
4. For each case in S, where $T_j$ executes a write(X) operation after $T_i$ executes a write(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
5. Test the precedence graph for the existence of cycle. If cycle is encountered, then schedule is not serializable, otherwise schedule is serializable.

Example:

Draw a precedence graph for the following schedule and identify whether it is conflict serializable or not. If it is conflict serializable, identify equivalent serial schedule also.

Schedule : $R_2(X)\ W_2(X)\ R_1(X)\ W_1(X)\ R_2(Y)\ R_3(X)\ W_2(Y)\ W_3(X)$

Sol:




| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | $R_2(X)$ | |
| | $W_2(X)$ | |
| $R_1(X)$ | | |
| $W_1(X)$ | | |
| | $R_2(Y)$ | |
| | | $R_3(X)$ |
| | $W_2(Y)$ | |
| | | $W_3(X)$ |

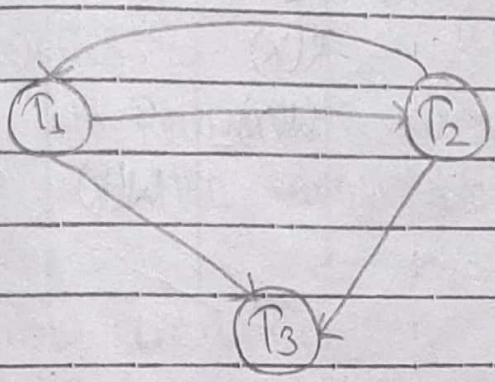
Since, the precedence graph doesnot have any cycle. Thus, the schedule is serializable. The equivalent serial schedule can be achieved if transaction operations are taken in order $T_2 \rightarrow T_1 \rightarrow T_3$ then the equivalent serial schedule is

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | $R_2(X)$ | |
| | $W_2(X)$ | |
| | $R_2(Y)$ | |
| | $W_2(Y)$ | |
| $R_1(X)$ | | |
| $W_1(X)$ | | |
| | | $R_3(X)$ |
| | | $W_3(X)$ |

Example:

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| R(X) | | |
| | W(X) | |
| | commit | |
| W(X) | | |
| commit | | W(X) |
| | | commit |

Sol?




## View Serializibility :-

- A schedule is view serializable if it is equivalent to some serial schedule.
- It is based on view equivalence.
- Every conflict serializable schedule is also a view serializable, however some view serializable schedule are not conflict serializable.
- Blind writes can occur in any view serializable schedule which is not conflict serializable.
- In computing blind writes occur when a transaction writes a value without reading it.

- To check for view serializibility of a schedule, we must create all possible combinations of the transactions.
- Thus determining whether a schedule is view-serializable is NP-complete problem.

Example:
Consider the following schedule.

$S_1$

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| R(X) | | |
| | W(X) | |
| W(X) | | |
| | | W(X) |

$S_2$

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| R(X) | | |
| W(X) | | |
| | W(X) | |
| | | W(X) |

The schedule $S_1$ is view serializable since it is view equivalent to serial schedule $S_2$.

## Schedule Based on Recoverability :-

- Serializable schedule ensures consistency of database during concurrent execution of transactions in multi-user environment.
- If there is no transaction failure occurs at the middle of the execution, serializable schedules work fine.
- But if transaction failure occurs at middle of the execution, we need to undo the effect of the transaction to ensure automicity property of the transaction.
- Further in case of system that allows concurrent execution,

if a transaction $T_i$ is failed and aborted then any transaction $T_j$ that is dependent upon $T_i$ must also be aborted.

- A transaction $T_j$ is said to be dependent upon $T_i$ if $T_j$ read the data item that is written by $T_i$.
- To address this issue we need to restrict the types of schedules that are acceptable from the database recovery point of view.
- On the basis of recoverability schedules can be divided into two categories:
    1) Recoverable schedule
    ii) Non-recoverable schedule.

1) Recoverable schedule :-

- A schedule S is said to be recoverable if for each pair of transaction $T_i$ and $T_j$ in S following condition is satisfied.
- Transaction $T_j$ must commit after transaction $T_i$, if $T_j$ reads the value of data item that is written by transaction $T_i$.

Example:

Consider the following two schedules which are exactly same except that first schedule shows succesful execution of transaction and second shows failed transactions.

$S_1$

| $T_1$ | $T_2$ |
|---|---|
| R(X) | |
| W(X) | |
| | R(X) |
| | W(X) |
| commit | |
| | commit |

$S_2$

| $T_1$ | $T_2$ |
|---|---|
| R(X) | |
| W(X) | |
| | R(X) |
| | W(X) |
| Abort | |
| | Abort |

- These schedules are recoverable.
- Schedule $S_1$ is recoverable because $T_1$ commits before $T_2$ that makes the value read by $T_2$ correct, then $T_2$ commit itself.
- In schedule $S_2$, if $T_1$ aborted, $T_2$ has to aborted because the value of data item X it reads is incorrect and $T_2$ also aborted.
- In both cases, the database is left in a consistent state.

## Non-recoverable schedule:-

When in any schedule if the committed transaction read the value of data item from any un-committed transaction then this type of schedule is said to be non-recoverable schedule.

Example :-

Consider the following schedule.

| $T_1$ | $T_2$ |
|---|---|
| R(X) | |
| W(X) | |
| | R(X) |
| | W(X) |
| | commit |
| Abort | |

- The schedule is unrecoverable, because $T_2$ read the value of the data item X written by $T_1$, and committed.
- $T_1$ later aborted, therefore the value of $T_2$ is wrong, since $T_2$ is already committed, it can't be aborted.
- Thus, this schedule is unrecoverable.

## Cascading and Cascaddess Schedules:-

### Cascading schedule:-

The phenomenon in which a single transaction failure leads to a series of transaction roll back is called cascading rollback and such schedules are called cascading schedules.

### Cascadeless schedule:-

- A schedule is cascadeless if transaction $T_j$ reads value of data item X only after transaction $T_i$ has written to data item X and terminated (i.e. either

committed or aborted) for each pair of transaction $T_i$ and $T_j$ in the schedule.

- Cascadeless schedule avoid cascading roll back by disallowing to read uncommited data, but this way may lead to serial schedule.

**Strict schedule:-**

A schedule is said to be strict if it satisfy following two condition for each pair of transaction $T_i$ and $T_j$.

i) Transaction $T_j$ reads value of data item X only after transaction $T_i$ has written to data item X and terminated (i.e. either committed or aborted).

ii) Transaction $T_j$ writes value of data item X only after transaction $T_i$ has written to data item X and terminated (i.e. either committed or aborted).

Example:

| $T_1$ | $T_2$ |
|---|---|
| R(X) | |
| R(Y) | |
| | R(Y) |
| W(X) | |
| commit | |
| | R(X) |
| | W(X) |
| | Abort |

[Fig:- Strict schedule]

# UNIT 3:
## Concurrency Control Techniques

### Concurrency Control Techniques:-

Those techniques that are used to control the interaction among concurrent transaction is called concurrency control techniques. The concurrency control techniques are required to control the interaction among the concurrent transactions. Objectives of concurrency control mechanism are listed as below:

i) To enforce isolation (through mutual exclusion) among conflicting transactions.
ii) To preserve database consistency through consistency preserving execution of transaction.
iii) To resolve read-write and write-write conflicts.

### Need of Concurrency Control:

If transactions are executed serially, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here are some typical examples:

1) Lost update problem:

This problem occurs when two transactions that access the same database item have their

operations interleaved in such a way that the value of data item written by one transaction is overlapped by another transaction and hence result to the incorrect value of the database.

Example:

Consider the two transactions try to update inventory of products $P_1$ simultaneously. Let initial value of $P_1 = 300$.

| $T_1$ | $T_2$ | value of data item |
|---|---|---|
| $R(P_1)$ | | $T_1 \cdot P_1 \leftarrow 300$ |
| $P_1 = P_1 + 100$ | | $T_1 \cdot P_1 \leftarrow 300 + 100 = 400$ |
| | $R(P_1)$ | $T_2 \cdot P_1 \leftarrow 300$ |
| | $P_1 = P_1 + 50$ | $T_2 \cdot P_1 \leftarrow 300 + 50 = 350$ |
| $W(P_1)$ | | $T_1 \cdot P_1 \leftarrow 400$ (writes) |
| | $W(P_1)$ | $T_2 \cdot P_1 \leftarrow 350$ (writes) |
| | commit | |
| commit | | |

ii) The dirty read (Temporary update) problem:

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value which causes transaction $T_2$ to have dirty value of the data item.

Example:

Consider the two transactions try to update inventory of products $P_1$ simultaneously. Let initial value of $P_1 = 300$.

| $T_1$ | $T_2$ | value of data item |
|---|---|---|
| $R(P_1)$ | | $T_1 \cdot P_1 \leftarrow 300$ |
| $P_1 = P_1 + 50$ | | $T_1 \cdot P_1 \leftarrow 300+50 = 350$ |
| $W(P_1)$ | | $T_1$ writes $P_1 \leftarrow$ ~~300~~ 350 |
| | $R(P_1)$ | $T_2 \cdot P_1 \leftarrow 350$ |
| | $P_1 = P_1 + 100$ | $T_2 \cdot P_1 \leftarrow 350+100 = 450$ |
| Abort | | $T_1$ is aborted (undo) |
| | $W(P_1)$ | $T_2$ writes $P_1 \leftarrow 450$ |
| | commit | |

iii) Incorrect Summary Problem:

If one transaction is calculating an aggregate summary functions on a number of records while other transaction is updating some of these records, the aggregate function may calculate some values before they are updated and other after they are updated results in incorrect summary.

Example:

Consider a schedule in which $T_1$ is trying to find sum of all units inventory while another transaction $T_2$ is trying to update inventory. Let initial value of $P_1$ and $P_2$ is 300 and 100 respectively and initial value of data item sum is 0.

| $T_1$ | $T_2$ | value of data item |
|---|---|---|
| $R(P_1)$ | | $T_1.P_1 \leftarrow 300$ |
| $sum = sum + P_1$ | | $T_1.sum \leftarrow 0 + 300 = 300$ |
| $R(P_2)$ | | $T_1.P_2 \leftarrow 100$ |
| $sum = sum + P_2$ | | $T_1.sum \leftarrow 300 + 100 = 400$ |
| commit | | $T_1$ committed |
| | $R(P_2)$ | $T_2.P_2 \leftarrow 100$ |
| | $P_2 = P_2 + 50$ | $T_2.P_2 \leftarrow 100 + 50 = 150$ |
| | $W(P_2)$ | $T_2.P_2 \leftarrow 150$ (writes) |
| | commit | $T_2$ is committed. |

The final sum is not correct because the value of $P_2$ is concurrently updated by the transaction $T_2$. Thus, $T_1$ unable to get the updated value of $P_2$. The ~~current~~ correct value of sum is 450.

iv) Unrepeatable read problem:

This problem occurs when a transaction T reads an items twice and the item is changed by another transaction T' between the two reads. Hence, T receives different values for its two reads of the same item.

for example:

Consider a scenario that may occur in a departmental store. Transaction $T_1$ checks inventory of product $P_2$ which is found to be 15. In the mean time, another transactions read inventory of products $P_2$ and decrement it by 10. Now, if $T_1$ reads inventory of

product $P_2$ again, it is unable to get previous value of $P_2$ and hence may lead to undesirable situations such as storing negative value in inventory of $P_2$. Assume the initial value of inventory of product $P_2$ is 15.

| $T_1$ | $T_2$ | value of data item |
|---|---|---|
| $R(P_2)$ | | $T_1 \cdot P_2 \leftarrow 15$ |
| | $R(P_2)$ | $T_2 \cdot P_2 \leftarrow 15$ |
| | $P_2 = P_2 - 10$ | $T_2 \cdot P_2 \leftarrow 15 - 10 = 5$ |
| | $W(P_2)$ | $T_2$ writes $P_2 \leftarrow 5$ |
| | commit | |
| $R(P_2)$ | | $T_1 \cdot P_2 \leftarrow 5$ |
| $P_2 = P_2 - 10$ | | $T_1 \cdot P_2 \leftarrow 15 - 10 = -5$ |
| $W(P_2)$ | | $T_1$ writes $P_2 \leftarrow -5$ |
| commit | | |

The negative value of $P_2$ may lead the database in unpredictable state.

concurrency is used to get efficiency.



2ta transaction le sang sangai data lai update garna napaos.

## The Concurrency Control Protocol :-

- In a multiprogramming environment where more than one transactions can be concurrently executed, there is a need of protocol to control the concurrency of transaction to ensure automicity and isolation properties of transaction.
- Concurrency control protocols can be broadly divided into four categories:
    1. Lock Based Protocols
    2. Time Stamp Based Protocols
    3. Validation Based Protocols
    4. Multi-version schemes.

## 1. Lock Based Protocols:- dakalai use garxa vane lock garxa and use gari sakesi unlock garinxa.

- One way to ensure serializibility is to access the data items in mutually exclusive manner i.e. while one transaction is accessing data item, no other transaction should be allowed to access the data item.
- Lock variables are most commonly used approach for achieving mutual exclusion.
- Locks are of two types:
    a) Binary locks
    b) Shared / Exclusive locks

### a) Binary Locks:-

- Binary lock is a variable that can be only in two states.
- It is either locked or unlocked.
- Normally, lock state is represented by 1 and unlock state is represented by 0.

- To use the simple binary scheme, every transaction must satisfy the following rules:

i) A transaction T must issue the operation lock (X) before performing any read (X) or write (X) operations.

ii) A transaction T must issue the operation unlock (X) after finishing all read (X) and write (X) operations.

iii) A transaction T will not issue a lock (X) operation if the data item X is already locked by it.

iv) A transaction T will not issue an unlock (X) operation if the data item (X) is not locked by it.

b) Shared / Exclusive locks:-

- This type of lock is also called multiple mode lock.
- It is a variable that can be in any of three states: unlocked, read-locked (shared-lock) and write-lock (exclusive-lock).

i) Shared-lock:-

If a transaction acquires shared-lock (read-lock) on data item X then other transaction can also acquire shared-lock on item X, but no transaction can acquire exclusive-lock (write-lock) on data item X.

ii)

ii) Exclusive lock :-

If a transaction acquires exclusive lock on data item X then no other transactions can acquire shared/ exclusive lock on the data item.

Lock compatibility matrix of shared/exclusive lock is as follows :

| $T_i$ \ $T_j$ | R(S) | write(X) |
|---|---|---|
| R(S) | True | False |
| W(X) | False | False |

Fig :- Lock compatibility matrix

Lock Conversion :-

- Sometimes, it is desirable to convert locks from one locked state to another.
- Such conversion of lock state is called lock conversion.
- Lock conversion can be of two types:

i) Lock upgrade :-

- This operation is used to convert existing read lock to write lock.
- Transition $T_i$ can upgrade read-lock on an item X only when no other transaction $T_j$ holds read-lock on the data item X.
- Otherwise lock upgrade operation will be rejected.

ii) Lock downgrade :-
- This operation is used to convert existing write lock to read lock.
- If a transaction holds write lock on data item X, no other transaction can have any lock on data item X.
- Therefore, transaction $T_i$ can always convert write-lock on data item X to read-lock.

## Two-phase locking protocol (2PL) :-
- The two phase locking protocol is used to ensure the serializibity in database.
- The protocol is implemented in two phase.
- Growing phase and shrinking phase.

i) Growing phase : lock lagaune kaam garxa.
- In this phase, we put read or write lock based on need on the data.
- In this phase we does not release any lock.
- Remember that all lock operation must preceede first unlock operation appeared in a transaction.

ii) Shrinking phase : lock release garne.
- This phase is just release of growing phase.
- In this phase we release read and write lock but does not put any lock on data.
- Unlock operations can only appear after last lock operation.
- For a transaction there two phase must be mutually exclusive.

- This means, during locking phase, unlocking phase must not start and during unlocking phase locking phase must not begin.

## Types of 2PL :-

Two phase locking protocol can be categorized into following four:

i) Basic 2PL (simple 2PL)
ii) Conservative 2PL.
iii) Strict 2PL
iv) Rigorous 2PL

### i) Basic 2PL :- chahesi lock garne ani sakesi release garda jane.

In this protocol, required data items are locked incrementally and once the operation with data item is finished locks are released from data items.

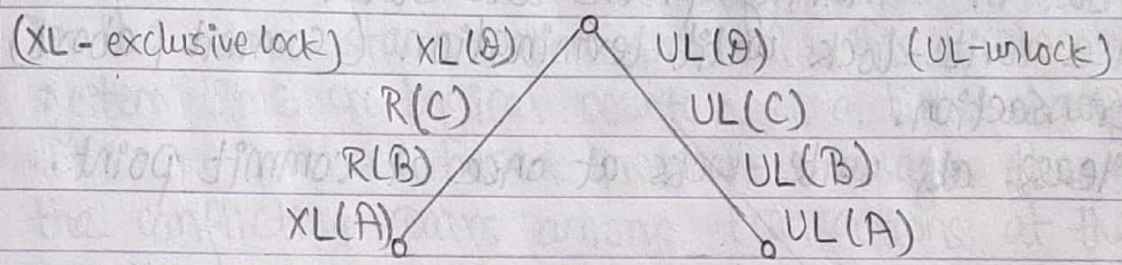



Fig :- Simple 2PL

Problems with simple 2PL:
- Unnecessary waiting
- Deadlock
- Cascading rollback

ii) Conservative 2PL :-

- This is also called static 2PL.
- It requires a transaction to lock all the items it accesses before the transactions begins execution by predeclaring it read and write sets.

Advantage:

- It guarantees deadlock free schedule.

Disadvantages:

- Unnecessary waiting
- Cascading rollback.

iii) Strict 2PL :-

- The growing phase of strict 2PL is same as the simple 2PL, but in shrinking phase the strict 2PL does not release write-lock until termination (commit / abort) of transaction.
- It release all write-locks at once at commit point.

Advantage:

- Cascading roll back free

Disadvantage:

- Deadlock may occur.

iv) Rigorous 2PL:-

- Rigorous 2PL is even stricter than strict 2PL.
- Here all locks (read and write locks) are held by transactions till termination (commit or abort).
- In this protocol transactions can be serialized in the order in which they commit.

Advantage:

- Cascading rollback free.

Disadvantage:

- Deadlock may occur.

2. Time stamp based concurrency control:-

The most commonly used concurrency protocol is the time stamp based protocol. This protocol uses either system time or logical counter as a time stamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas time stamp based protocol start working as soon as a transaction is created.

. Time stamp:-

Time stamp is a unique fixed value, denoted by TS($T_i$) which we associate when a transaction enters into a system. This time stamp is assigned by the database system before the transaction $T_i$ starts execution. If a transaction $T_i$

has been assigned time stamp $TS(T_i)$, and a new transaction $T_j$ enters the system, then $TS(T_i) < TS(T_j)$.

There are two simple methods for implementing this scheme:

1. Use the value of the system clock as the time-stamp i.e. a transaction's timestamp is equal to the value of the clock when the transaction enters the system.

2. Use a logical counter that is incremental after a new time stamp has been assigned.

To implement timestamp protocol, we associate with each data item x, two timestamp values:

i. Write-timestamp(x): denotes the largest timestamp of any transaction that execute write(x) successfully.

ii. Read-timestamp(x): denotes the largest timestamp of any transaction that execute read(x) successfully.

The timestamp must have following two properties:

* Uniqueness: ensures that no equal time stamp values can exist.

* Monotonicity: ensures that time stamp values always increase.

## Basic Timestamp Ordering Protocol:

Timestamp ordering protocol ensures serializibility among transaction in their conflicting read and write operations. This protocol ensures that any conflicting read and write operation are executed in timestamp order. This protocol operate as follows:

Note:

* The timestamp of transaction $T_i$ is denoted as $TS(T_i)$.
* Read timestamp of data item X is denoted by R-timestamp(X).
* Write timestamp of data item X is denoted by W-timestamp(X).

i) When a transaction $T_i$ issues a read(X) operation

```
if (W-timestamp(X) > TS(Ti))
{
    abort and rollback T and reject the operation;
}
else
{
    read(X)
    R-timestamp(X) = TS(T);
}
```

ii) When a transaction $T_i$ issues a write(X) operation:

```
if (R-timestamp(X) > TS(Ti) or, W-timestamp(X) > TS(Ti))
{
    abort and rollback T and reject the operation;
}
else
{
    write(X);
    W-timestamp(X) = TS(Ti);
}
```

Example:

Assume timestamps of $T_1$ and $T_2$ is 100 and 110 respectively and initial value of X is 500.

| $T_1$ | $T_2$ | Time stamp |
|---|---|---|
| Read(X) | | R-timestamp ← 100 |
| | Read(X) | R-timestamp ← 110 |
| X = X+200 | | |
| | X = X+200 | |
| | Write(X) | W-timestamp(X) ← 110 |
| Write(X) | | W-timestamp(X) > TS($T_1$) and |
| Abort $T_1$ | | hence time stamp ordering violated. |

$T_2$ le read or write grna $T_1$ should be aborted or commited

**Strict timestamp ordering:-**

Strict timestamp ordering is a variation of basic timestamp ordering that ensures that the schedules are strict and conflict serializable. In this variation, a transaction T, that issues a read(X) or write(x) operation such that TS(T) > W-timestamp(X) has its read or write operation delayed until the transaction $T_1$ that wrote the value of X has committed or aborted. Rest of the working of strict I/O protocol is same as the basic I/O protocol. To implement this algorithm locking is required. This algorithm does not cause deadlock, since T waits for $T_1$ only if TS(T) > TS($T_1$).

Example:

Assume timestamp of $T_1$ and $T_2$ is 100 and 110 respectively and initial value of X is 500.

| $T_1$ | $T_2$ | Timestamp |
|---|---|---|
| Read (X) | | R-timestamp (X) ← 100 |
| | Read (X) | R-timestamp (X) ← 110 |
| | X = X+200 | |
| X = X+200 | | |
| Write (X) | | W-timestamp (X) ← 100 |
| | Write (X) | TS($T_2$) > W-timestamp (X), therefore $T_2$ waits until $T_1$ is terminated. |

## Thomas Write Rule:-

Thomas write rule (TWR) is a modification to the basic timestamp ordering protocol. It relaxes conflict serializibility and provides greater concurrency by reflecting obsolete write operation. In this rule, read operation remain unchanged but write operation are different from the basic timestamp-ordering protocol.

- When a transaction $T_i$ issues a write (X) operation

```
if (TS(Ti) < R-timestamp(X))
{
  abort and rollback transaction T and reject operation
}
if (TS(Ti) < W-timestamp(X))
{
  ignore this write but continue transaction
}
```

```
else
  {
    write(X)
    W-timestamp = TS(Ti)
  }
```

Example:
Assume timestamp of $T_1$ and $T_2$ is 100 and 110 respectively and initial value of X is 500. Then,

| $T_1$ | $T_2$ | Timestamp |
|---|---|---|
| Read(X) | | R-timestamp(X) ← 100 |
| | Write(X) | W-timestamp(X) ← 110 |
| Write(X) | | TS($T_1$) < W-timestamp(X), therefore from TWR write(X) operation must be ignored. |
| | | |
| | | |

Thomas write rule allows some view serializable schedules that are not conflict serializable.

## Multiversion concurrency control:

In multiversion database systems, each write operation on data item, say 'x' create a new version of X. When a Read(X) operation is issued, the system select one of the version of X to read. The concurrency control scheme must ensure that the selection of the version to be read is done in manner that ensures serializibility. There are two multi-version schemes:

a) Multi-version techniques based on timestamp ordering
b) Multi-version technique based on two-phase locking.

## a) Multi-version technique based on timestamp ordering:

In this technique, several versions $X_1, X_2, X_3, \ldots, X_k$ of each data item X are kept by the system. For each version the value of the version $X_k$ and the following timestamp are kept:

- current is the value of version $X_k$
- W-time stamp ($X_k$) is the timestamp of the transaction that created version $X_k$.
- R-time stamp ($X_k$) is the largest timestamp of any transaction that successfully read version $X_k$.

The scheme operates as follows when transaction $T_i$ issues a read(X) or write(X) operation. Let $X_k$ denoted the version of X whose time stamp is the largest write timestamp less than or equal to $TS(T_i)$. This mean $X_k$ is the version of data item X that is written by youngest transaction that is older than transaction $T_i$.

i. If transaction $T_i$ issues a Read(X) operation, then the value returned is the content of version $X_K$.

ii. If a transaction $T_i$ issues a write(X) operation, and if $TS(T_i) < R\text{-timestamp}(X_K)$, then transaction $T_i$ is rolled back. otherwise, if $TS(T_i) = W\text{-timestamp}(X_K)$ then contents of $X_K$ are over written, otherwise a new version of X is created.

Example 1:
Consider the following transaction schedule.

| $T_1$ | $T_2$ | $T_3$ | Description |
|---|---|---|---|
| Read(X) | | | Read version $X_0$ |
| Write(X) | | | Create new version $X_1$ |
| | Read(X) | | Reads $X_1$ |
| | | Read(X) | Reads $X_1$ |
| | Write(X) | | Rollback $T_2$, since $TS(T_2) <$ |
| | | Write(X) | R-timestamp(X). |

Example 2:

| $T_1$ | $T_2$ | $T_3$ | Description |
|---|---|---|---|
| Read(X) | | | Read version $X_0$ |
| Write(X) | | | Creates new version $X_1$ |
| | Read(X) | | Read $X_1$ |
| Read(Y) | | | Read $Y_0$ |
| Write(X) | | | Overwrites $X_1$, since $TS(T_1) = W$- |
| | Write(X) | | timestamp($X_1$) creates version $X_2$. |
| | | Read(X) | Read $X_2$ |
| | | Write(X) | Create version $X_3$ |

## Multi-version Two-phase locking using certify locks:

The multiversion two-phase locking protocol attempt to combine the advantages of multiversion concurrency control with the advantages of two-phase locking. In this scheme, there are three locking modes for an item read, write and certify. Hence, the state of lock(X) for an item X can be one of the Read-locked, write-locked, certify-locked or unlocked.

Lock held by a transaction

| Lock Requested | Read | Write | Certify |
|---|---|---|---|
| Read (Shared) | Yes | Yes | No |
| Write (Exclusive) | Yes | No | No |
| Certify | No | No | No |

Fig: Lock compatibility matrix.

This scheme works as follows:

- Multiversion 2PL allows other transactions to read data item X while a transaction is performing write(X) operation.
- This is accomplished by allowing two versions for each item X, one version must always have been written by some committed transaction and the second version of X is created when a transaction T acquires a write lock on the item, say this version X'.
- Thus, other transaction can write the value of X' as needed without affecting the value of the committed version X.
- Once a T is ready to commit, it must obtain a certify lock on all items that are written by it before it can commit.

- The certify locks are not compatible with read lock, so the transaction may have to delay its commit until all its write-locked items are released by any reading transaction in order to obtain the certify locks.
- Once the certify locks acquired, the committed version X of the data item is set to the value of version X', version X' is discarded and the certify locks are then released.

Example:

Consider the following transaction schedule.

| $T_1$ | $T_2$ | Description |
|---|---|---|
| RL(X) | | |
| R(X), X=X-n | | |
| WL(X) | | |
| W(X) | | Creates new version X' of X |
| | RL(X) | |
| | R(X) | Read original version X |
| | sum = sum+X | |
| Certify-lock(X) | | Waits until $T_2$ release locks on X |
| | Unlock(X) | |
| | Commit | |
| Unlock(X) | | |
| Commit | | Set X=X' |

## Validation (Optimistic) concurrency control schemes:

In optimistic / validation concurrency control technique, no checking is done while the transaction is executing. In this scheme, updates in the transaction are not applied directly to the database items until the transaction reaches its end. During transaction execution, all of updates are applied to local copies of the data items that are kept for the transaction. At the end of transaction execution, a validation phase checks whether any of the transactions updates violate serializibility. If serializibility is not violated, the transaction is committed and the database update is updated from the local copies. Otherwise the transaction is aborted and then restarted later.

The validation protocol requires that each transaction requires two or three different phases. Those phases are:

i. Read Phase: A transaction is activated and reads committed values of data items from the database and puts those values in local variable. (no tests), (just to sutai data read garxa)

ii. Validation Phase: In this phase checking is performed to ensure that serializibility is not violated, if the transaction updates are applied. schedule valid xa ki nai vanne check garxa.

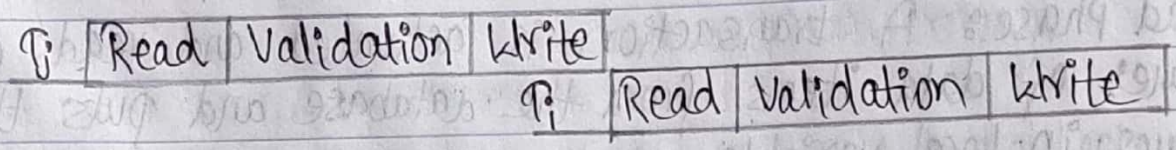
iii. Write phase: If the validation phase is successful, the transaction updates are applied to the database in write phase, otherwise the updates are discarded and the transaction is restarted.

To perform the validation test, we need to know when the various phases of transaction took place. Therefore, three different timestamp to each transactions is associated.

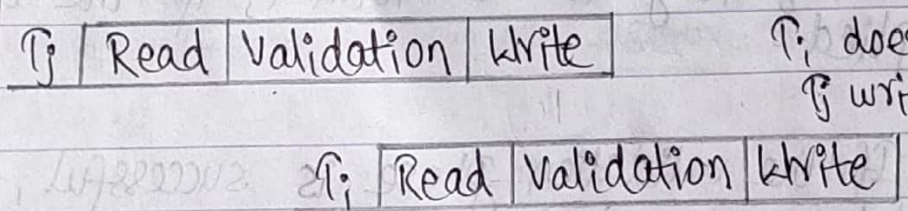
i. Start($T_i$): the time when $T_i$ starts execution.
ii. Validation($T_i$): the time when $T_i$ finished its read phase and started its validation phase.
iii. Finish($T_i$): the time when $T_i$ finished its write phase.

The validation test for transaction $T_i$ requires that, for all transactions $T_j$ with $TS(T_j) < TS(T_i)$, one of the following condition must hold:

a. $T_j$ completes its write phase before $T_i$ starts its read phase.




b. $T_i$ starts its write phase after $T_j$ completes its write phase and the read set of $T_i$ and write set of $T_j$ are disjoint.



c. Both the read set and write set of $T_i$ have no items in common with the write set of $T_j$ and $T_j$ completes its read phase before $T_i$ completes its read phase.

| $T_j$ | Read | Validation | Write |
|---|---|---|---|

| $T_i$ | Read | Validation | Write |
|---|---|---|---|

$T_i$ does not read or write anything that $T_j$ writes.

## Graph Based Protocol:

A graph based protocol consist of following rules:

1. Only lock-X is allowed.
2. First lock can be acquired on any data item.
3. Subsequent locks are allowed only if the parent is locked.
4. Data item may be unlocked at any time.
5. Each data item can be accessed at most once.
6. Relocking by same transaction is not allowed.

All schedule that are legal under the tree protocol are conflict serializable.

Example:

Given the following tree-structured database graph.

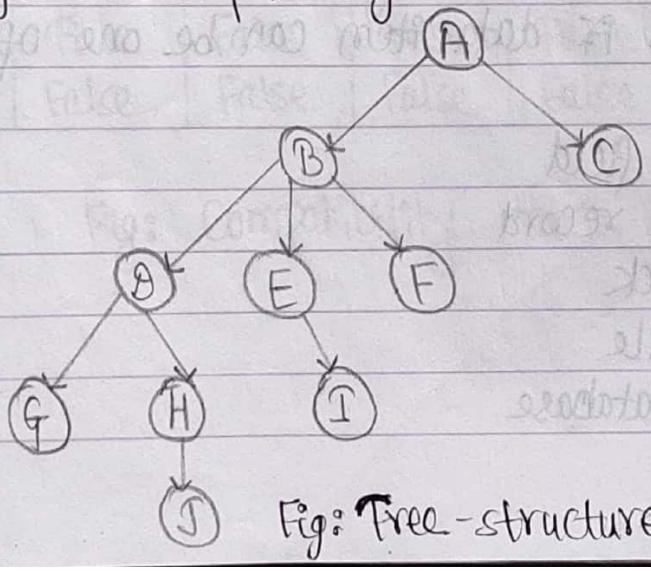



Fig: Tree-structured database graph

The serializable schedule based on the above database graph is:

| $T_1$ | $T_2$ |
|---|---|
| Lock-X(B) | |
| | Lock-X(D) |
| | Lock-X(H) |
| | Unlock(D) |
| Lock-X(E) | |
| Lock-X(D) | |
| Unlock(B) | |
| Unlock(E) | |

Fig: serializable schedule under the tree protocol.

## Multiple Granularity:-

The size of database item is often called the item granularity. To develop a concurrency control scheme which allow more level of concurrency we need a mechanism to allow the system to define multiple levels of granularity. This is done by allowing data items to be of various sizes and defining a hierarchy of data granularities, where the small granularities are nested within larger ones. A data item can be one of the following:

1. Database field
2. Database record
3. Disk block
4. Whole file
5. Whole database

To implement the multiple granularity scheme we use new locking scheme called intension lock modes along with previously used shared and exclusive lock. There are three types of intension mode locks.

i. Intension-shared (IS) mode: Explicit locking at lower level of tree but only with shared lock. child mode (exclusive & shared) both lock laguna painxa. & exclusive lock

ii. Intension-exclusive (IX) mode: Explicit locking at lower level with exclusive or shared lock. only shared lock. only

iii. Shared and Intension-Exclusive (SIX): Subtree rooted by that node is locked explicitly in shared mode and explicit locking is done at a lower level with exclusive mode.

The multiple-granularity locking protocol uses these lock modes to ensure serializibility. The compatibility matrix of the lock is given below:

| | IS | IX | S | SIX | X |
|---|---|---|---|---|---|
| IS | True | True | True | True | False |
| IX | True | True | False | False | False |
| S | True | False | True | False | False |
| SIX | True | False | False | False | False |
| X | False | False | False | False | False. |

Fig: Compatibility matrix

A transaction $T_i$ that attemps to lock on a node Q must follow these rules:

1. Must observe the lock compatibility.
2. Root of the tree must be locked first and it can be locked in any mode.
3. A node Q can be locked by $T_i$ in S or IS mode only if the parent of Q is currently locked by $T_i$ in either IX or IS mode.
4. A node Q can be locked by $T_i$ in X, SIX, or IX mode only if the parent of Q is currently locked by $T_i$ in either IX or SIX mode.
5. $T_i$ can lock a node only if it has not unlocked any node so far.
6. $T_i$ can unlock a node Q only if none of the child of Q is currently locked by $T_i$

**Deadlock :-**

A system is in a deadlock state if there exist a set of transaction such that every transaction in the set is waiting for data item that is locked by another transaction in the set. Deadlock is the main problem associated with two phase locking protocol. Only conservative 2PL free from deadlock.

Example:

| $T_1$ | $T_2$ |
|---|---|
| Lock(X) | |
| Read(X) | |
| | Lock(Y) |
| | Read(Y) |
| Lock(Y) | |
| (wait for Y) | |
| | Lock(X) |
| | (wait for X) |

In this example above, both transaction need data item x and y to continue further. Transaction $T_1$ waits for y and transaction $T_2$ waits for x. This situation is called deadlock.

## Dealing with deadlock:-

There are two principle methods for dealing with the deadlock problem:

i) Deadlock prevention
ii) Deadlock detection and recovery

### 1) Deadlock prevention:-

This protocol is used to ensure that the system will never enter a deadlock state. There are two approaches of deadlock prevention.

a) Wait-die scheme
b) Wound-wait scheme

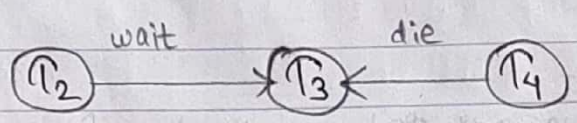
resource line wala die hunna

a) Wait-die scheme: younger transaction le older transaction lai wait gnna paunna..

- It is a non-preemptive technique for deadlock prevention.
- When transaction $T_i$ requests a data item currently held by $T_j$, $T_i$ is allowed to wait only if it has a timestamp smaller than that of $T_j$, otherwise $T_i$ is rolled-back (dies).

If $TS(T_i) < TS(T_j)$ then
    $T_i$ waits
else
    $T_i$ dies

Example:
Suppose that transaction $T_2, T_3, T_4$ have timestamp 5, 10, 15 respectively. If $T_2$ requests a data item held by $T_3$, then $T_2$ will wait. If $T_4$ request a data item held by $T_3$, then $T_4$ will be rolled back.



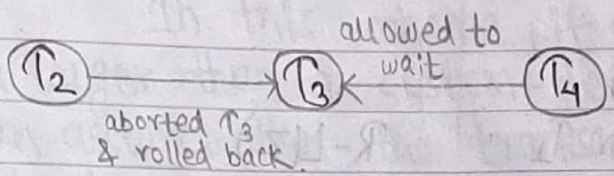

resource line wala die hunxa.

b) Wound-wait scheme:

- It is a preemptive technique for deadlock prevention.
- When a transaction $T_i$ requests a data item currently held by transaction $T_j$, $T_i$ is allowed to wait only if it has a timestamp larger than that of $T_j$, otherwise $T_j$ is rolled back ($T_j$ is wounded by $T_i$).
- This scheme prefer older transactions.

If $TS(T_i) < TS(T_j)$ then
    $T_j$ is wounded (rolled back)
else
    $T_i$ waits

Example:
In above example, if transaction $T_2$ requests a data item held by transaction $T_3$, then the data item will be preempted from $T_3$ and $T_3$ will be aborted and rolled back. If $T_4$ requests a data item held by $T_3$, then $T_4$ will wait.




ii) **Deadlock detection:**

Deadlock detection checks whether deadlock state actually exist in the system before taking any actions. A deadlock can be detected using wait-for graph of schedule. A deadlock is said to occur if there is a circular path in wait-for graph.

The wait-for graph consist of a pair $G = (V, E)$ where V is a set of vertices which represents all the transaction in the system. E is the set of edges where each element is an ordered pair $T_i \rightarrow T_j$.

Example:

Identify whether the following schedule is deadlock free or not with the help of wait-for graph.

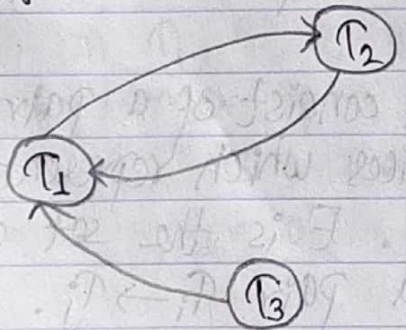
$S_1 = R_1(X), R_2(Y), W_1(X), R_2(X), R_3(Z), W_3(Z), R_1(Y), R_3(X), W_1(Y)$.

Sol^n:

We can rewrite the above schedule with lock and unlock.

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| R-L(X) | | |
| R(X) | R-L(Y) | |
| | R(Y) | |
| W-L(X) | | |
| W(X) | | |
| | R-L(X) | |
| | Wait for $T_1$ | R-L(Z) |
| R-L(Y) | | R(Z) |
| R(Y) | | W-L(Z) |
| W-L(Y) wait for $T_2$ | | W(Z) |
| | | R-L(X) wait for $T_1$ |

The above schedule can be represented in the wait-for graph as:




since, the wait-for graph contains cycle, the given schedule suffers from deadlock problem.

Fig: Wait-for graph of above schedule.

## Recovery from Deadlock:-

Once deadlock is detected we need some mechanism to resolve deadlock situation. Victim selection and timeout mechanism are two simple techniques used to recover from deadlocked condition.

1. Victim selection:

In this approach one of the transaction is selected and rolled back. Such a transaction is called victim transaction.

11. Timeout mechanism:

In this scheme, if a transaction waits for a period longer than a system-defined time out period, the system assumes that the transaction may be deadlocked and abort it, regardless of whether a deadlock actually exists or not.

## Starvation:-

When we use lock based protocol for concurrency control, there is another problem called starvation. It occurs when a transaction cannot proceed for a definite period of time. While other transaction in the system continue normally. This may occur if the waiting scheme for locked item is unfair, giving priority to some transaction over other. One solution for starvation is giving a first come first serve queue. Another method is to increase the priority of a transaction if it waits longer.

# Recovery System

## Failure Classification:

There are various types of failure that may occur in a system, each of which needs to be deal within a different manner. Following are some types of failure that occur in the database system.

1. **Transaction failure:**

There are two types of error that may cause a transaction to fail.

i. **Logical error:**

The transaction can no longer continue with its normal execution because of some internal condition such as bad input, data not found, overflow, or resource limit exceeded.

ii. **System error:**

The system has entered in an undesirable state, as a result of which a transaction cannot continue with its normal execution.

2. **System Crash:**

There is a hardware malfunction that causes the loss of the content of volatile storage and brings transaction processing to a halt.

3. Disk failure:

A disk block loses it content as a result of either a head crash or failure during a data transfer operation. Copies of data on other disks, or archival backup on ternary media are used to recover from the failure.

To recover from any failure at first we need to identify mode of failure and then identify the algorithm. These algorithm are known as recovery algorithm and these algorithm consist of two parts.

i. Action taken during normal transaction processing to ensure that enough information exists to allow recovery from failure.

ii. Action taken after a failure to recover the database contents to a state that ensures database consistency, transaction automicity and durability.

Storage Structure:

The storage media can be classified based on speed, capacity and resiliance to failure. There are three categories of storage media:

i. Volatile storage:

Volatile storage doesnot usually survive information after system crashes. Example of those media are main memory and cache memory.

ii. Non-volatile storage:
Information residing in non-volatile storage survive even though system is crashed. Example of these media are magnetic disk and tapes.

iii. Stable storage:
Information residing in stable storage is never lost. Although stable storage is theoritically impossible to obtain, but it can be closely approximately by using different techniques.

## Stable storage implementation:

To implement stable storage, we need to replicate the needed information in several non-volatile storage media with independent failure mode. To update the information we need to use some technique to ensure that failure during data transfer does not damage the needed information.

## Recovery concept:

The recovery manager of a DBMS is responsible for ensuring two important properties of transaction: automicity and durability. It ensures automicity by undoing the actions of transaction that do not commit and durability by making sure that all actions of committed transaction survive system crashes and media failures. Recovery from transaction failures usually

means that the database is restored to the most recent consistent state just before the time of failure. To do this, the system must keep information about the changes that were applied to data items by the various transactions. This information is typically kept in the system log.

There are two types of recovery techniques, which can help DBMS in recovering as well as maintaining the automicity and durability of transactions:

i. Log Based Recovery:

This strategy maintains the logs of each transaction and writing them onto some stable storage before actually modifying the database. Such file in stable storage is called log file. This log file is used for recovery process after transaction failure occurs.

ii. Shadow Paging:

This strategy maintains two versions of database in disk at different places. One version represents consistent state that is achieved after previous committed transaction which cannot be updated by current transaction. Another version is disk copy of the database that can be updated by current transaction.

## Need of Database Recovery:

A computer system, like any other devices, is subjected to failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room, even sabotage. In any failure information may be lost. Some reasons of database failure are discussed below:

i. System crash (Computer failure):
A hardware or s/w error occurs in the computer system during transaction execution. If the hardware crashes, the content of the computer's internal memory may be lost.

ii. A transaction or system error:
Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

iii. Local error or Exception conditions:
Certain conditions necessitate cancelation of the transaction. For example, data for the transaction may not be found.

iv. Concurrency control / enforcement:
The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializibility or because several transaction are in a state of deadlock.

v. Disk failure:
Some disk block may lose their data because of read or write malfunction or because of a disk read/write head crash. This may happen during a read or write operation of the transaction.

vi. Physical problems and catastropher:
This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, overwriting disks or tapes by mistakes, and mounting of a wrong tape by the operator.

## Log Based Recovery:-

The most widely used structure for recording database modification is the log. The log is a sequence of log records and maintains a history of all update activities in the database. It is important that the logs are written prior to actual modification and stored on a stable storage. There are four types of log records that are recorded by recovery manager in log file.

i. Start record:

This record indicates start of a transaction and contains transaction identifier and keyword start.

Syntax:-

$< T_i, start >$

where, $T_i$ is transaction identifier.

ii. Update record:

This is recorded when a transaction perform write(X) operation. It contains four fields: transaction identifier, data item identifier, old value and new value of data item.

Syntax:-

$< T_i, X, OV, NV)$

where,

$T_i$ = transaction identifier
X = data item
OV = old value
NV = new value

iii. Commit record:

This is recorded when a transaction is committed. It contains two fields: transaction identifier and keyword commit.

Syntax:-

$< T_i, commit >$

where, $T_i$ is identifier.

iv. Abort record:

This is recorded when a transaction is aborted. It contains two fields: transaction identifier and keyword abort.

Syntax:-

<$T_i$, abort>

where, $T_i$ is transaction identifier.

Example:

A simple schedule and corresponding log file recorded by recovery manager during execution of the schedule is shown below:

| $T_1$ | $T_2$ |
|---|---|
| Read(X) | |
| X = X-50 | |
| Write(X) | |
| | Read(Z) |
| | Z=Z-100 |
| | Write(Z) |
| Read(Y) | |
| Y=Y+50 | |
| Write(Y) | |
| Commit | |
| | Commit |

Fig: Schedules

| $T_{id}$ | Operation | Data item | Old value | New-value |
|---|---|---|---|---|
| $T_1$ | start | | | |
| $T_1$ | Update | X | 200 | 150 |
| $T_2$ | start | | | |
| $T_2$ | Update | Z | 300 | 200 |
| $T_1$ | Update | Y | 300 | 350 |
| $T_1$ | commit | | | |
| $T_2$ | commit | | | |

Fig: Log recorded for schedule S

There are two main techniques of log based database recovery:

i) Differed Update (No-Undo / Redo) Policy
ii) Immediate Update (Undo / Redo) Policy

## Buffer Management (Data buffering and Updating):

- When a transaction starts, the data required for that transaction is loaded to main memory from disk.
- The blocks residing in disk blocks are called physical block and the blocks residing in main memory is called buffer block.
- Each block is associated with two bits:

* Modified bit: If the value of buffer data is modified then bit is set 1, otherwise 0.

* Pin-unpin bit: If the value of pin-unpin bit is 1, then buffer block cannot be written back to disk as yet. Otherwise, buffer block can be written back to disk.

When the value of modified bit is 1, then we need to write back the block into disk. There are two methods of flushing modified buffer blocks into disk.

1. In-place updating:
   It writes buffer block to the same original disk location.

ii. Shadow updating:
   It writes buffer block to the disk location that is different from original.

When the modified buffer blocks are written back to disk, following policies are used:

i. Steal/No-steal policy:
   Steal policy allows buffer manager to write a buffer block to disk before transaction commits. And the no steal policy does not allow buffer manager to write a buffer block back to disk before transaction commits.

ii. Force/No force policy:
   Force policy ensures that all the pages updated by transaction are immediately written to disk when the transaction commit. The no-force policy does not require

force policy: commit vayesi lekhnai par x9
No-force policy: commit vayesi lekhna pardaina

to write changes. For frequently changed objects, a no-force policy allows updates to be merged and so reduce the number of write operations.

Combining above policies used in writing back buffer block to disk block, there are four different ways for handling recovery:

i) steal / No-force (Undo/Redo)
ii) steal / Force (Undo / No-redo)
iii) No-steal / No-force (Redo / No-undo)
iv) No-steal / Force (No-undo / No-redo)

## Check pointing:

In log based recovery, the log file is consulted by recovery manager to identify the transaction that need to be redone the transaction that need to be undone after transaction failure. To do this, the buffer manager need to search all log files. Most modern DBMS use the concept of check point.

Check pointing is a mechanism where all the previous logs are removed from the buffer and stored permanently in storage disk. Check point declares a point before which the DBMS was in consistent state and all the transaction were committed. The following steps define a checkpoints operations.

i. Suspend execution of transaction
ii. Force write modification buffer data to disk
iii. Write a record to the log, save the log to disk
iv. Resume normal transaction execution.

## Write Ahead Logging:

In this approach, data pages can't be written to disk until the log records describing those changes are written to disk first. If the data page written before the log record, the data page creates modification on the disk that can't be rolled back if the transaction fails before the log record is written to disk. WAL states that:

For undo: Before the data file is written to disk the log must first save to the disk.

For redo: Before the transaction executes its commit operation, all its changes must be written to the log and the log must be saved on a stable storage.

## Deferred Update Recovery Policy:

These techniques defer or postpone any actual updates to the database until the transaction reaches it commit point. During transaction execution, the updates are written to the log file. After the transaction reaches its commit point, the log file is force-written to disk, then the updates are recorded in the database. If the transaction fails before reaching its commit point, there is no need to undo any operations because the transaction has not affected the database on disk in any way. Recovery technique based on deferred update are therefore known as no-undo/redo techniques. Redo is needed in the case the system fails after a transaction commit but before all its changes

are recorded on disk. In this case, the transaction operations are redone from the log files.

i. Recovery using deferred update in single-user environment:
Uses a two list of transaction: the committed transaction since the last check points and the active transaction (at most one transaction is fall in this category because the system is single user).

ii. Recovery using deferred update in a multi-user environment:
These techniques are closely related with the concurrency control techniques. This environment requires some concurrency control mechanism to guarantee isolation property of transactions.

## Immediate Update Recovery Policy:

In this technique, when a transaction issues a update command, the database can be updated immediately, without any need to wait for the transaction to reach its commit point. However, this technique record any changes to the database to the log before it applied to the database, so that we can recover in the case of failure. If a transaction fails after recording some changes in the database must be undone, that is, rolled back.

On the other hand, if force policy of writing is used then redo operation is not needed. If no-force writing policy is used, redo operation is needed for committed transaction.

i. Immediate update in multi-user environment:
It uses strict 2PL so that none of the locks are released till transaction is committed. It uses two list of transactions: committed and active transactions since the last check point.

ii. Immediate update in single user environment:
It uses two list of transactions by the system; the committed transactions since the last checkpoint and the active transaction (at most one transaction will fall in this category, because the system is single-user).

⇒ Shadow Paging (Already discussed)

Failure with loss of non-volatile storage:
The basic scheme is to dump the entire content of the database to stable memory periodically. No transaction can be active during the dump procedure. To recover from the loss of non-volatile memory, we restore the database from the archive and all the transaction that have been committed since the most recent dump are redone. This is also known as an archival dump. Dumps of the database and check pointing are very similar.