Database Management System (CSC260)

Note By:

- **Roshan Bist**
- SNSC, Mahendranagar
- https://www.facebook.com/roshanbist.roshanbist.3
- https://www.facebook.com/notejunction/
- notejunction360@gmail.com

❤️

If my note really helps you, then you can support me on eSewa for my hard work.

eSewa ID: 9806470952

Unit-1

Database and Database Users:-

Data → Data are the raw facts that are found after some experiment, observation or after some experiments. Data itself do not provide any meaning but after processing it becomes information.

Database → The collection of data organized in some specific manner is known as database. For example, university database for maintaining information about students, courses and grades in university.

Database Management System → The database, its processing methods and the set of rules and conditions to be followed is collectively known as database management system (DBMS). It is the way to store data and also provides mechanism for mainupulation of data. It is basically just a computerized record-keeping system. The primary goal of DBMS is to store and retrive data in both convinent and efficient manner. Oracle, MySQL, SQL-Server, MS Access etc. are some examples of DBMS.

⊛ Applications of DBMS:

- Banking → To store information about customers, their account number, balance etc.
- Airlines → For reservations and schedule information.
- Telecommunication → To keep records of customers, call made, balance left, generating monthly bills etc.
- Sales → To keep information of customers, products list, purchase information etc.
- Universities → To keep record of students, courses, marks of students etc.
- Human Resources → To keep record of employee, their ~~salary~~ salery etc.
- Manufacturing → To store orders, tracking production of items etc.

⊛. Characteristics of the Database Approach:

In past traditional file processing approach was used in which each user defines and implements the files needed for a specific software application as part of programming the application. But in database approach, a single repository maintains data that is defined once and then accessed by various users repeatedly through queries, transactions and application programs. The main characteristics of the database approach versus the file-processing approach are as follows:-

i) Self-describing nature of a database system:

The database system contains not only the database itself but also a complete description. The definition is stored in the DBMS catalog which contains information such as structure of each file, the type and storage format of each data item. The information stored in the catalog is called meta-data and it describes the structure of the primary database.

The newer types of database systems do not require meta-data. Rather the data is stored as self-describing data that includes the data item names and data values together in one structure. These database systems are known as NOSQL systems.

ii) Insulation between programs and data, and data abstraction:

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access the file. But DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog seperately from the access programs. We call this property program-data-independence.

iii) Support of Multiple Views of the Data:

A database typically has many types of users, each of whom may require a different perespective or view of the database. A view may be a subset of the database or it may contain virtual data. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views. For example one user of database may be interested only in accessing and printing the transcript and a second user who is interested only in checking that students have taken all the prerequisites of each course.

iv) Sharing of data and multiuser transaction processing:

A DBMS must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The database must include concurrency control software to ensure that several users trying to update the same data.

DBMS should ensure that concurrent transactions operate correctly and efficiently. These types of applications are generally called online transaction processing (OLTP) applications. The concept of ~~at~~ a transaction has become central to many database applications. A transaction is an executing program or process that includes one or more database accesses such as reading or updating of database records.

⊛. Actors on the Scene:-

For a small personal database such as the list of addresses, typically one person defines, constructs and manipulates the database and there is no sharing. But in large organizations many people are involved in the design, use and maintenance of a large database with hundreds or thousands of users. The people whose jobs involve the day-to-day use of a large database are called actors on the scene.

i) Database Administrators → In database environment the primary resource is database itself and secondry resource is the DBMS and related software. Administrating these resources is the responsibility of the database administrator (DBA). The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is also accountable for problems such as security breaches and poor system response time.

ii) Database Designers → Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirement and to create a design that meets the requirements. The database design must be capable of supporting the requirements of all user groups.

iii) End Users → End users are the people whose jobs require access to the database for querying, updating and generating reports. There are several categories of end users;

- Causal end users → They occasionally access the database, but they may need different information each time. They use a sophisticated database query interface to specify their requests.
- Navie end users → They constantly query and update the database using standard types of queries and updates called canned transactions that have been carefully programmed and tested.
- Sophisticated end users → It includes engineers, scientists, business analysists and others in order to meet their complex requirements.

• Standalone users → They maintain personal databases by using ready-made, program packages that provide easy-to-use menu-based or graphics-based interfaces.

iv) System Analysists and Application Programmers (Software Engineers):

System analysists determine the requirement of end users specially navie and parametric end users and develop specifications for standard canned transactions that meet the requirements. Application programmers implement these specifications as programs then they test, debug, document and maintain these canned transactions.

⊛ Workers Behind the Scene:

Those who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job are called workers behind the scene. They include following categories:

i) DBMS system designers and implementors → They design and implement the DBMS modules and interfaces as a software package. A DBMS is very complex software consisting many components including modules for implementing catalog, query language processing, interface processing, handling data recovery and security etc. The DBMS must interface with other system software such as operating system and compilers for various programming languages.

ii) Tool developers → They design and implement tools which are software packages that facilitate database modeling and design and improve performance. Tools are optional packages that are often purchased seperately. In many cases, independent software vendors develop and market these tools.

iii) Operators and maintenance personal → They are also called system administration personnel and are responsible for the actual running and maintenance of the hardware and software environment for the database system.

⊛. [redacted] Advantages of using the DBMS Approach:-

The DBA must utilize the different capabilities to accomplish a variety of objectives related to the design, administration and use of a large multiuser database. Following are some advantages of using DBMS and the capabilities that a good DBMS should possess.

i) Controlling Redundancy:- The file based management systems contained multiple files that were stored in many different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy.

This is prevented in database as there is a single database and any change in it is reflected immediately. Because of this, there is no chance of encountering duplicate data.

ii) Restricting unauthorized access:- Data Security is vital concept in a database. Only authorised users should be allowed to access the database and their identity should be authenticated using a username and password. Unauthorised users should not be allowed to access the database under any circumstances as it violates the integrity constraints.

iii) Providing Persistant Storage:- Programming languages typically have complex data structures such as struct or class definitions in C++ or Java. The values of program variables or objects are discarded once a program terminates unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage. When need arises to read this data, once more the programmer must convert the file.

The DBMS software is compatible with these programming languages such as C++ and Java and automatically performs necessary conversions. Hence a complex object can be stored in an object-oriented DBMS. Such an object is said to be persistent, since it survives the termination of program execution and can be later directly retrived by another program.

iv) Providing Backup and Recovery:- The backup and recovery subsystem of DBMS is responsible for recovery. For example, if the computer fails in the middle of a complex update transaction, the recovery system is responsible for making sure that the database is restored to the state it was in before the transaction started executing. Disk backup is also necessary in case of a catastrophic disk failure.

v) Providing Multiple User Interfaces:- Many types of users with varying levels of technical knowledge use a database, so a DBMS should provide a variety of user interfaces. These includes apps for mobile users, query language for causal users, programming language interfaces for application programmers, forms and commands for parametric users, and menu-driven interfaces and natural language interface for standalone users. Both forms-style interfaces and menu-driven interfaces are commonly known as graphical user interfaces (GUIs).

vi) Flexibility:- It maybe necessary to change the structure of a database as requirements change. For example, a new user group may emerge that needs information necessary to add a file to the database or to extend the data elements in an existing file. Modern DBMSs allow certain ~~database without~~ types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs.

vii) Availability of Up-to-Date Information → As soon as users update is applied to the database, all other users can immediately see that update. This availability of up-to-date information is essential for many transaction-processing applications such as reservation systems or banking databases.

viii) Economics of Scale:- The DBMS approach reduces the amount of wasteful overlap between activities of data-processing personnel in different projects or departments. This enables the whole organization to invest in more powerful processors, storage devices, rather than having each department purchase its own low performance equipment. This reduces overall costs of operation and management.

⊛. Purpose of DBMS / Why DBMS is required?

Traditionally, file processing system was used to manage information. It used to store data in various files of different application programs to extract or insert data to appropriate file.

File processing system has several drawbacks due to which DBMS is required. DBMS removes the problems found in file processing system. Some major problems of file processing system are:

i) Data redundancy and inconsistency.
ii) Difficult in accessing data.
iii) Data isolation.
iv) Integrity problem.
v) Security problem
vi) Atomicity problem etc.

Unit-2

Database System - Concepts and Architecture

⊛. **Data abstraction** → It refers to hiding mechanism of details of data organization and storage and the highlighting of the essential features for an improved understanding of data. To provide data abstraction is one of the fundamental characteristic of database approach. There are three levels of abstraction.

i) Physical level / lowest level → describes how data is actually stored in database.
ii) Logical level / Second lowest level → describes type of data stored and relation between them.
iii) View level / highest level → describes interaction between the users and system.

⊛. **Data Model:** Data model is a collection of concepts that can be used to describe structure of a database. It provides the necessary means to achieve the abstraction. It is a conceptual tool for describing data and relation among data. There are several data models which can be put into different categories:-

1) **Object-based logical models:**

Object-based logical model describe data at the logical and view levels. It has flexible structuring capabilities. It is of two types:

ⓐ. **Entity-Relationship model:** E-R model describes the design of database in terms of entities and relationship among them. An entity is a 'object' in real world with a set of attributes.

Eg. attributes: customer_id, customer_name, customer_address etc.

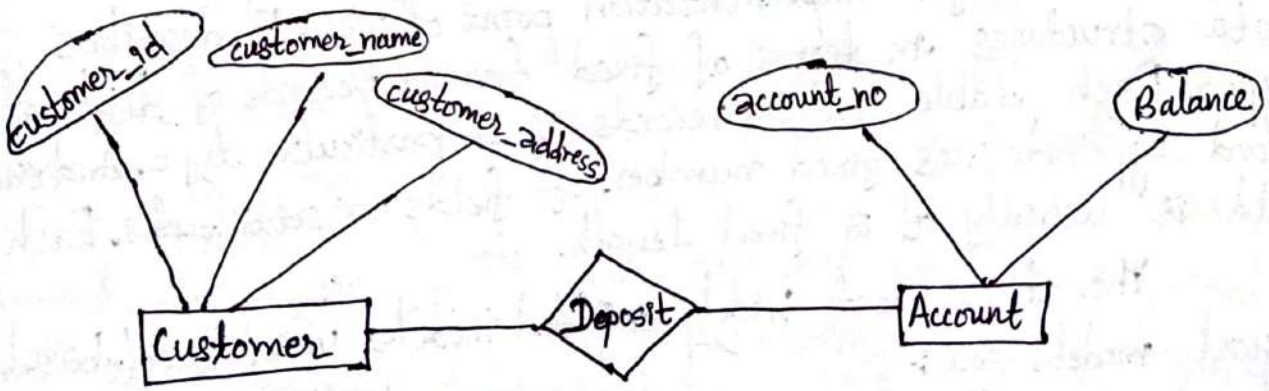


Fig. Sample of E-R model. (ER diagram)

An entity is a thing or object in real world which are described in database as a set of attributes. A relationship is an association among several entities. Logical structure of a database can be expressed graphically by ER diagram which include following components:

→ Rectangles (represent entity sets).
→ Ellipses (represent attributes)
→ Diamonds (represent relationship sets among entity sets)
→ Lines (link attributes to entity sets and entity sets to relationship sets).

(b). Object-Oriented Model:

The object oriented model is based on a collection of objects like the E-R model. Object contains values and bodies of codes which are called methods. An object can contain another object to an arbitarily deep level of nesting. Another feature of object-oriented model is inheritance i.e, new class can be derived from the existing class.

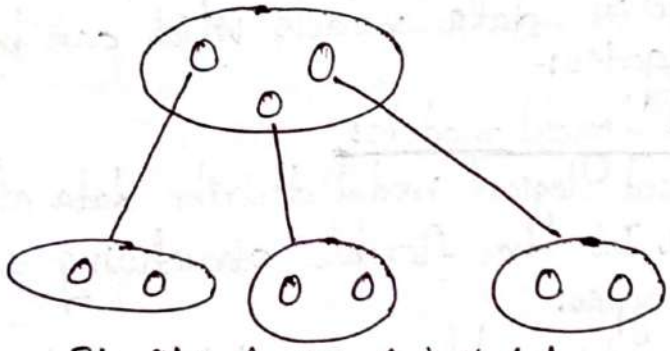


Fig. Object-Oriented Model

2) Record-based Logical Model:

Record-based logical model also describes data at logical and view level but it describes logical structure of database in more detail for implementation point of view. It describes data structures in terms of fixed format records of different types. Each table contains records of a particular type. And each record type defines fixed number of fields or attributes. Each field is usually of a fixed length.

The three most widely accepted models under record-based logical models are:-

ⓐ. Hierarchical model: In hierarchical data model the data is organized into a tree-like structure. The structure allows repeating information using parent/child relationships; each parent can have many children, but each child only has one parent.

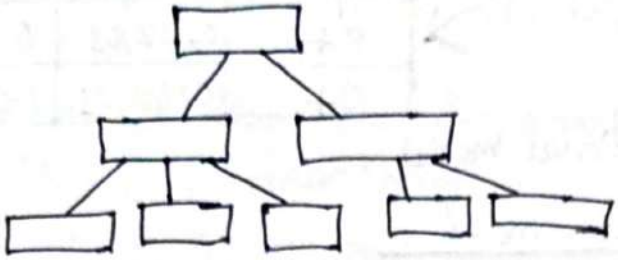


Fig. Structure of Hierarchical model

ⓑ. Network model: The data in network model are represented by collection of records and relationships among data are represented by links, which ~~are~~ can be viewed as pointers. The records in the database are organised as collection of arbitary groups. The main advantage of this model is its representation of relationships between entities is implemented using pointers which allows representation of aribitary relationship.

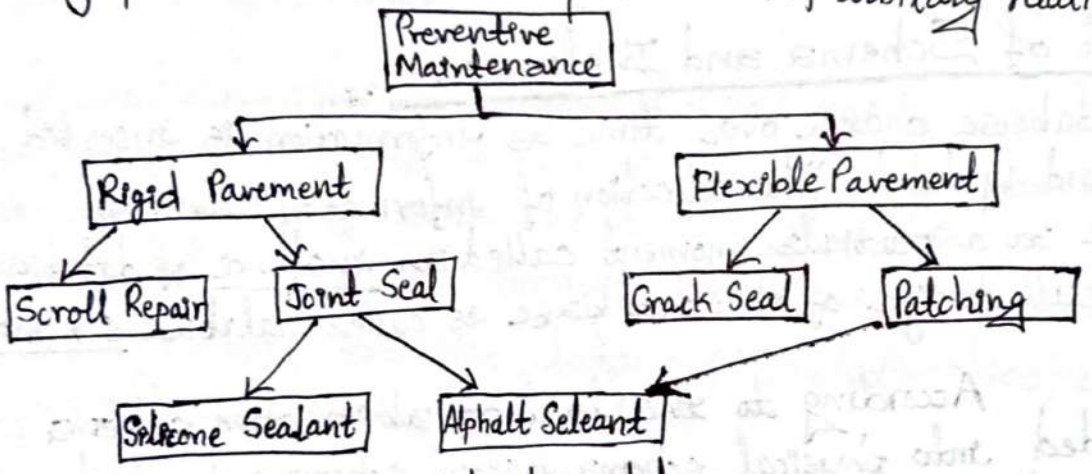


Fig. Network model

ⓒ. Relational model: The relational model uses a collection of tables and relationship among those tables. Tables have multiple columns and each column has a unique name. The main advantage of this model is its ability to represent data in a simplified format. The process of ~~mainup~~ manipulating record is simplified with use of certain key attributes used to retrive data.

Activity code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

key=24

Activity code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

Fig. Relational model

3) Self-describing data models:-

Self-describing data model is a semi-structured data model. The data stored in this model is generally associated with a scheme that is contained within the data property known as self-describing property. This model is referred to as self-describing because it not only contains the database itself but also meta-data which defines and describes the data and relationships between the tables in the database.

⊛. Concept of Schema and Instance:

Database change over time as information is inserted, deleted and updated. The collection of information stored in database at a particular moment called an instance of database. The overall design of the database is called database schema.

According to the level of abstraction schema are divided into physical schema, logical schema and subschemas. The physical schema describes the database design at the physical level. Logical schema describes database design at logical level. Database system may have several schemas. at the view level, it is called subschemas (can be query). It describes different views of database.

Logical schema is more important for the development of applications programs. Programmer constructs applications by using logical schema. The physical schema is hidden under the logical schema and it can change without affecting application programs.

⊛ Three-Schema Architecture and Data Independence:

Among many of the characteristics of database approach three most important characteristics listed below specify an architecture for database systems called three-schema architecture that was proposed to help active and visualize three characteristics:

i) Use of catalog to store the database description so as to mak it self-describing.

ii) Insulation of programs and data.

iii) Support of multiple user views.

The goal of three-schema architecture is to seperate the user applications from the physical database.

a) Internal level → The internal level has an internal schema, which describes the physical structure of database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

b) Conceptual level → The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations and constraints.

c) External or view level → It includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user groups.

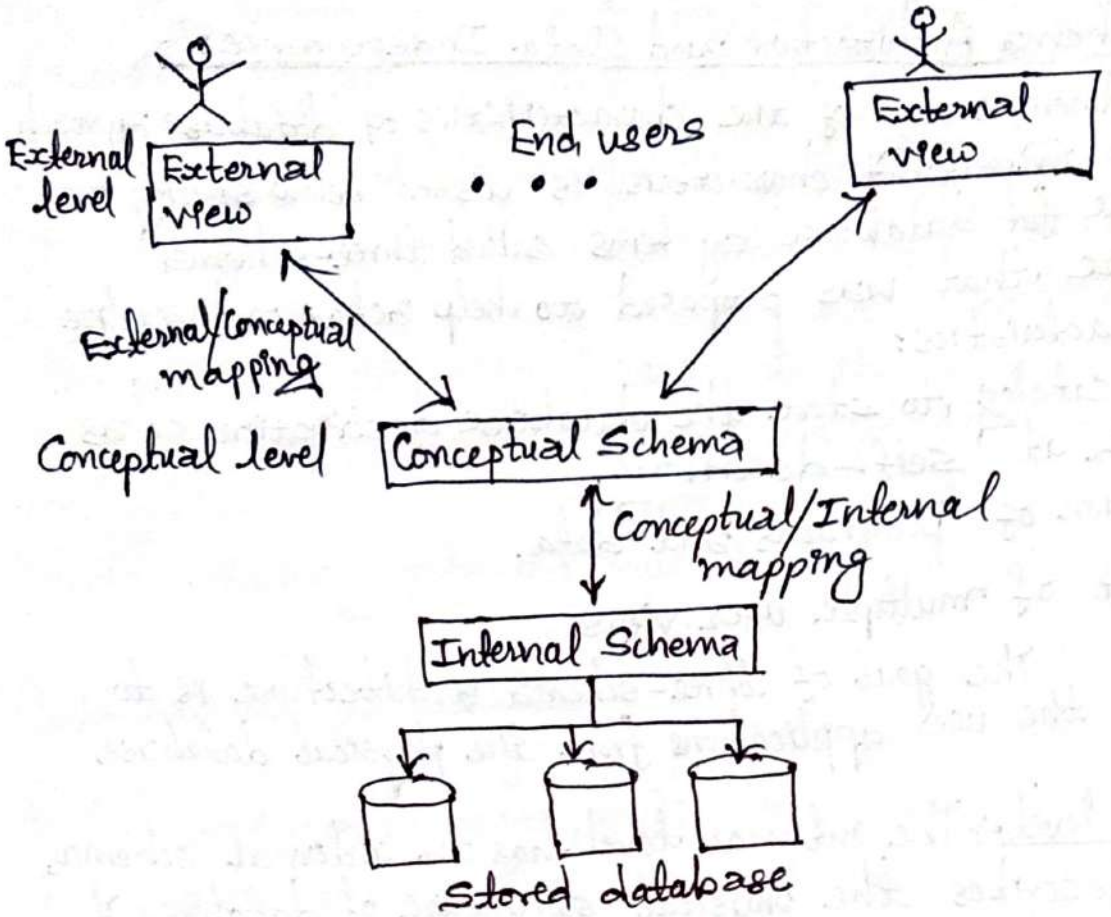


Fig. Three-Schema architecture.

The three-schema architecture is a convinent tool with which the user can visualize the schema levels in a database system. Most DBMSs do not seperate the three levels completely and explicitly but they support the three-schema architecture to some extent. The three-schemas are only descriptions of data the actual data is stored at physical level only. The process of transforming requests and results between levels are called mappings.

Data Independence:

The capacity to change the schema at one level of database system without having to change the schema at the next higher level is called data independence. Following are the two types of data independence:

1) Logical data independence → It is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the constraints, or to reduce database. Only the view definition

and the mappings need to be changed in a DBMS that supports logical data independence.

ii) Physical data independence → It is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need to be changed as well. Changes to the internal schema may be needed because some physical files were ~~recog~~ reorganized. If the same data as before remains in the database, we should not have to change the conceptual schema.

⊛. Database Languages:-

Database system provides following languages:-

ⓐ Data Definition Language (DDL):- This language is used to define data structures and specially database schemas. These statements are used to create, alter or drop data structures. ALTER, CREATE, DROP are some examples of DDL. It also allows to define storage structure and access methods for database system.

ⓑ. Storage Definition Language (SDL):- This language is used to define internal schema. It defines that what will be the physical structure of database like how many bites per field will be used, what will be the order of fields and how records will be accessed etc.

ⓒ View Definition Language (VDL):- This language is used to specify user views and their mapping to conceptual schema. It defines the subset of records available to classes of users. It creates virtual tables and the view appears to users like conceptual level. It specifies user interfaces.

(d). Data Manipulation Language (DML):- It is used at conceptual level and external level and is used to perform operations like Query, Delete Update or Insert. Read Only Queries are also sometimes considered as component of DML. It modifies ~~that d~~ the data but not schema or database objects. This language is further divided into following two types:-

i) Procedural DML → It is also called low-level DML. It simply contains series of computational steps to be carried out. Any procedure might be called at any point during program execution. In this user required to specify what data are needed and how they get those data.

ii) Non-Procedural DML → It is also called high-level or declarative DML. It describes the logic of computation without describing its control flow. In this user only required to what data needed without specifying how to get those data.

(*). DBMS Interfaces:

User-friendly interfaces provided by a DBMS may include following:-

(a) Menu-based Interface for Web Clients or Browsing:- These interfaces present the user with list of options called menus that lead the user through formulation of a request. Pull-down menus are a very popular technique in Web-based user interfaces. They are also often used in browsing interfaces which allow user to look through contents of a database.

(b). Apps for Mobile Devices:- These interfaces present mobile users with access to their data. For example banking, reservations and insurance companies provide apps that allow users to access their data through a mobile phone or mobile device. The apps have built-in programmed interfaces that typically allow users to login using their name and password.

ⓒ Forms-based Interfaces:- A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data. Forms are usually designed and programmed for navie users as interfaces to canned transactions.

ⓓ Graphical User Interfaces:- A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms.

ⓔ Natural Language Interfaces:- These interfaces accept requests written in English or some other language and attempt to understand them. The natural language interface refers to the words in its schema, as well as to the set of standard words in it's dictionary that are used to interpret the request.

ⓕ. Keyword-based Data Search:- These are somewhat similar to web search engines which accept strings of natural language words and match them with documents at specific sites or web pages. They use predefined indexes on words and use ranking functions to retrive and present resulting documents in a decreasing degree of match.

⊛. Database System Environment:-

A database environment is a collective system of components that comprise and regulates the group of data, management and use of data which consist of hardware, people & techniques of handling database.

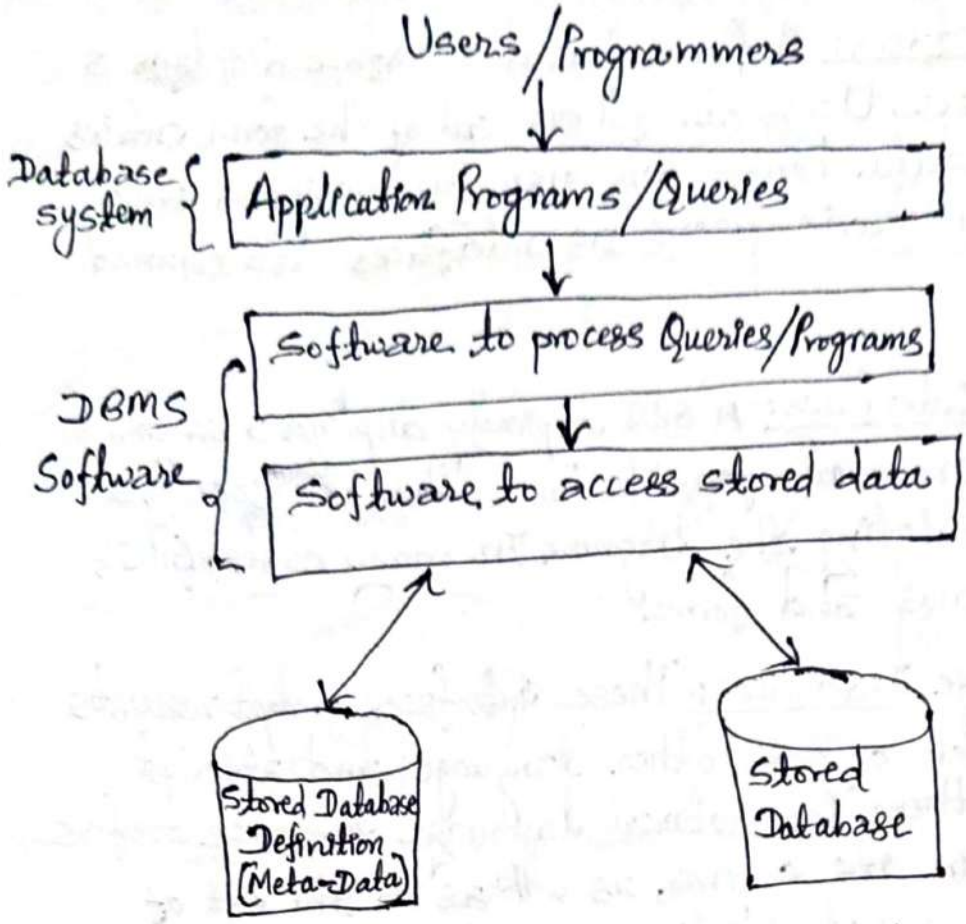


Fig. Database System Environment.

Here the hardware in a database environment means the computers and computer peripherals that are being used to manage a database, and the software means the whole thing right from the operating system (OS) to the application programs that include database management software like M.S. Access or SQL Servers. Again the people in the database environment include those people who administrate and use the system. The techniques are the rules concepts and instructions given to both the people and the software along with the data with the group of facts and information positioned within the database environment.

⊛ Centralized Database Systems:-

In centralized system, all programs run on the main host computer, including the DBMS, the application that accesses the database and the communication facilities that send and receive data from the users terminals. The users access the database through either locally connected or dial-up (remote) terminals. The terminals are generally dumb, having little or no processing

power of their own and consists of only a screen, keyboard and hardware to communicate with the host.

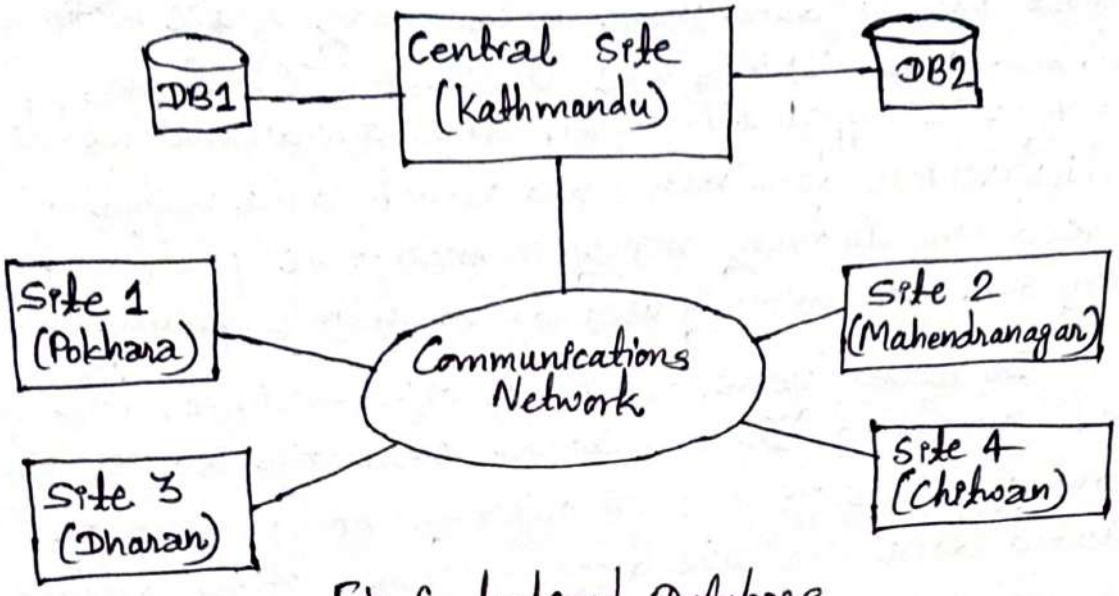


Fig. Centralized Database

⊛. Client/Server Database System:-

In a generalized concept, client PC is the computer from where the user requests for data and information and the server provides the requested information. The database application on the client PC referred to as the 'front end system' that handles all the screen and user input/output processing. The 'back end system' on the database server handles data processing and disk access.

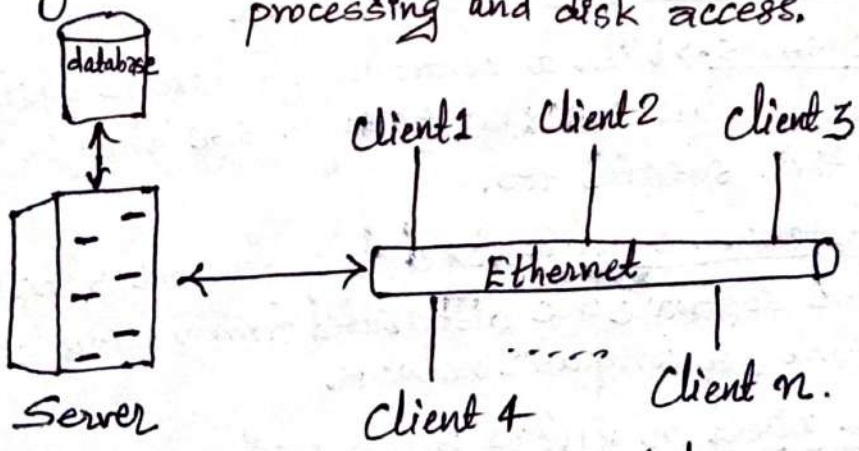


Fig. Client server database system.

Client server approach is implemented by two approaches: two-tier architecture and three-tier architecture.

⊛. Classification of Database Management Systems:-

ⓐ Classification based on data model:

The most popular data model in use today is the relational data model. Well-known DBMSs like Oracle, MS SQL Server, DB2 and My SQL support this model. Other traditional models such as hierarchical data models and network data models are still used in industry mainly on mainframe platforms. However, they are not commonly used due to their complexity.

In recent years, the newer object-oriented data models were introduced. In this model information is represented in the form of objects as used in object-oriented programming. Object-oriented databases are different from relational databases, which are table-oriented.

ⓑ. Classification Based on user numbers:

A DBMS can be ~~ca~~ classified based on the number of users it supports. It can be a single-user database system, which supports one user at a time, or a multiuser database system, which supports multiple users concurrently.

ⓒ Classification based on number of sites:

i) Centralized systems → With a centralized database system, the DBMS and database are stored at a single site that is used by several other systems too.

ii) Distributed database system → In this system the actual database and the DBMS software are distributed from various sites that are connected by a computer network.

ⓓ. Classification based on type of access path:

i) Homogenous distributed database systems → They use the same DBMS software for multiple sites. Data exchange between these various sites can be handled easily.

ii) Heterogenous distributed database systems → In this system different sites might use different DBMS software, but there is additional common software to support data exchange between these sites.

Unit-3

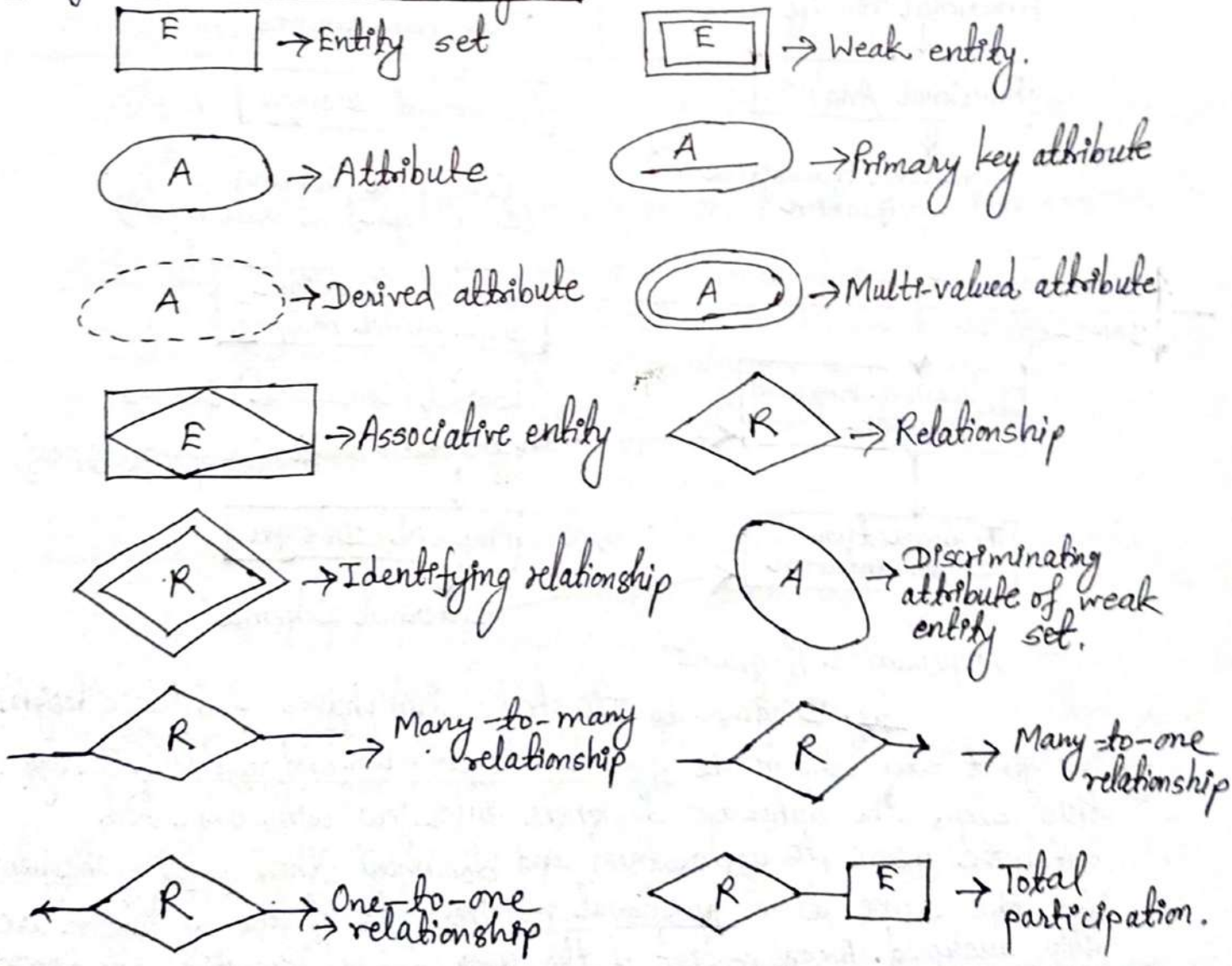
Data Modeling Using the Entity-Relational Model

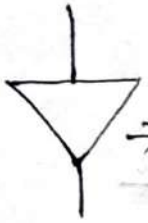
⊛ Introduction to ER-Diagram:

The E-R data model is based on a perception of real world that consist of a collection of basic objects called entities and relation among these objects. In an E-R model a database can be modeled as collection of entities, and relationship among entities.

Once the entity types, relationships types and their corresponding attributes have been identified the next step is to graphically represent these components using E-R diagram. E-R diagram is a graphical tool that demonstrates the relationships among various entities of database. It is used to design overall logical structure of database. While designing E-R diagrams, the emphasis is on the schema of database and not on the instances.

⊛ Symbols used in E-R diagram





→ Total generalization

→ ISA (specialization or generalization).

⊛ Concept of Conceptual Design: Using High-Level Conceptual Data Models for Database Design:

The below figure shows simplified overview of the database design process:-

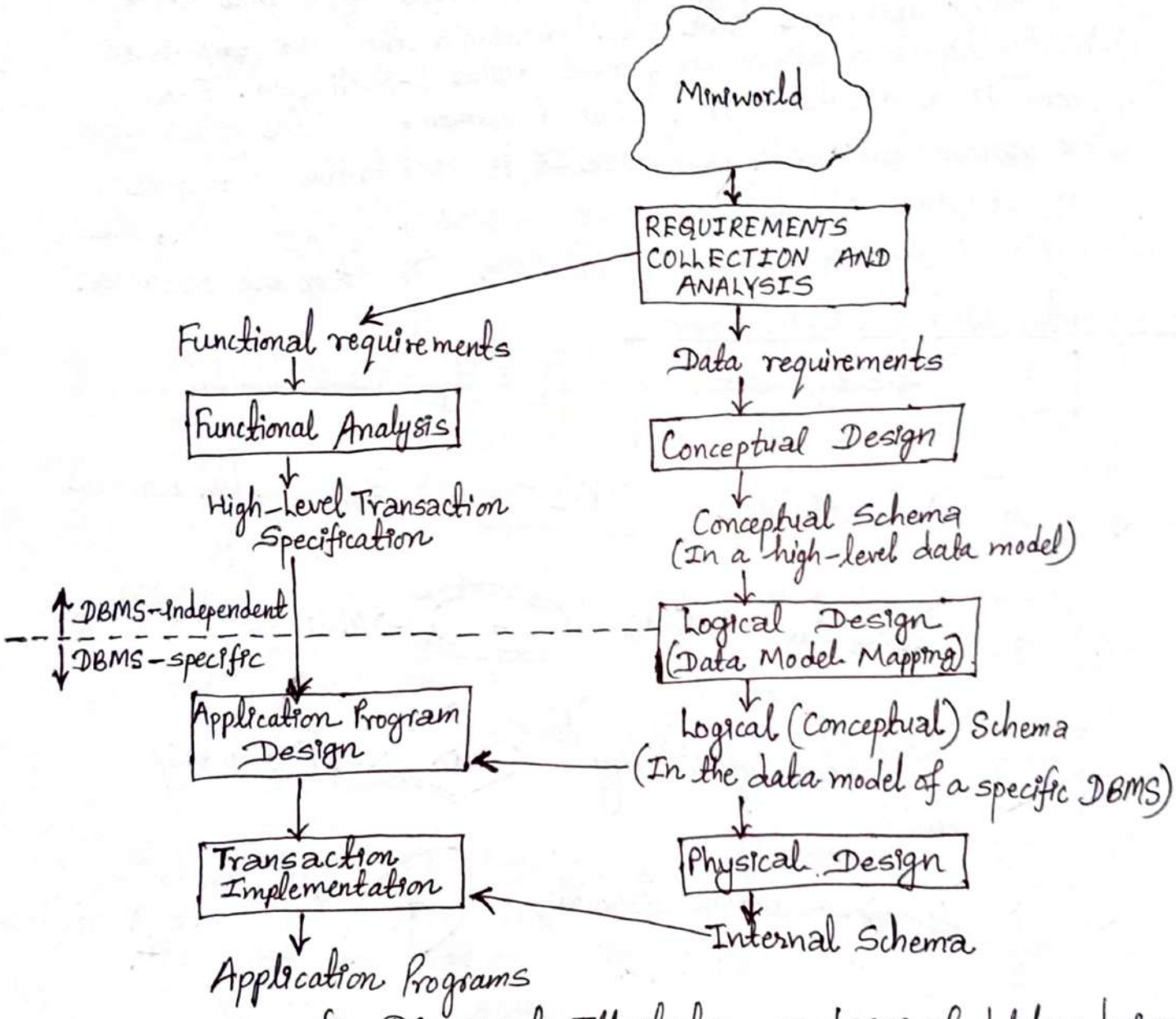


fig. Diagram to Illustrate main phases of database design.

→ The first step shown is requirements collection and analysis. During this step, the database designers interview with expected database users to understand and document their data requirements. At the same time functional requirements of the application are also analyzed. These consist of the user defined operations or transactions that will be applied to database.

→ Once the requirements have been collected and analyzed, the next step is to create <u>conceptual schema</u> for the database, using high-level conceptual data model. This step is called <u>conceptual design</u>. This includes descriptions of the data requirements of the users and includes descriptions of the entity types, relationships and constraints.

→ The next step in database design is the actual implementation of the database, using a commercial DBMS. (for e.g. SQL). In this step conceptual schema is transformed from the high-level data model into the implementation data model. This step is called <u>logical design</u>.

→ The last step is the <u>physical design</u>, during which the internal storage structures, file organizations, indexes, access paths and physical design parameters for database files are specified.

⊛ Entity Types, Entity Sets, Attributes, and Keys:-

<u>Entity</u> → Entity is a thing or object in the real world with an independent existence. An entity may be an object with a physical existence or it may be an object with a conceptual existence.

<u>Attributes</u> → The particular properties that describe entity are known as attributes. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salery, job etc. A particular entity will have value for each of its attributes.

<u>Entity Type</u> → Entity type defines a collection (or set) of entities that have same attributes. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has its own value(s) for each attribute.

<u>Entity set</u> → The collection of entities of a particular entity type in the database at any point in time is called an <u>entity set</u>. The entity set is usually referred to use the same name as the entity type, even though they are two seperate

concepts. For example, EMPLOYEE refers to both a type of entity as well as the current collection of all employee entities in the database.

Key → An entity attribute has value which is distinct for each individual entity, such a value is called key. It is an important constraint on the entities of an entity type. An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set, such an attribute is called a key attribute.

⊛. Concept of relationship types and relationship sets, roles and constraints:

A relation type R among n entity types $E_1, E_2, \ldots, E_n$ defines a set of associations or a relationship set among entities from entity types. Similar to the case of entity types and entity sets, a relationship type and its corresponding relationship set are referred by same name R. Mathematically, the relationship set R is a set of relationship instances r_i, where each r_i associates n individual entities $(e_1, e_2, \ldots, e_n)$, and each entity e_j in r_i is a member of entity set E_j, $1 \le j \le n$. Hence, a relationship set can be defined as a subset of the Cartesian product of the entity sets $E_1 \times E_2 \times \ldots \times E_n$.

Degree of relationship type → The degree of relationship type is the number of participating entity types. A relationship type of degree two is called binary and degree three is called ternary.

Role names and Recursive Relationships → Each entity type that participates in a relationship type ~~plas~~ plays a particular role in the relationship. The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and it helps to explain what the relationship means.

In some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for

distinguishing the meaning of the role that each participating entity plays. Such relationship types are called recursive relationships and self-referencing relationships.

Constraints on Binary Relationship Types →

There are two main types of binary relationship constraints: cardinality ratio and participation.

i) Cardinality Ratios for Binary relationships → The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate.

ii) Participation constraint → The participation constraint specifies whether the existence of an entity depends on its being related to another entity through the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in and is sometimes called the minimum cardinality constraint.

(4). Concept of Weak Entity Types and Partial Keys:-

Entity types that do not have key attributes of their own are called weak entity types. An entity type should have a key attribute which uniquely identifies each entity in entity set, but there exists some entity type for which key attribute can't be defined. These are called weak entity type. The entity sets which do not have sufficient attributes to form a primary key are known as weak entity sets, and the entity sets which have a primary key are known as strong entity sets. As the weak entities do not have any primary key, they cannot be identified on their own, so they depend on other entity known as owner entity. Weak entities always has total participation but strong entity may not have total participation. Weak entities are represented with double rectangular box in the ER diagram and the identifying relationships are represented with double diamond. Example: The existence of rooms is entirely dependent on existence of hotel. So room can be seen as weak entity of hotel.

Partial key: The set of attributes that are used to uniquely identify a weak entity set is called the partial key. The partial key of the weak entity set is also known as discriminator. It is just a part of the key ~~as~~ as only a subset of the attributes can be identified using it. It is partially unique and can be combined with other strong entity set to uniquely identify the tuples. ~~For exam~~ Partial keys are represented by dashed line in ER diagram as shown in figure below. For example:- Let us take a apartment as weak entity and building as a strong entity type. Now, the apartment number is not globally unique i.e, more than one apartment may have same number globally but it is unique for particular building. Thus apartment number is a partial key.

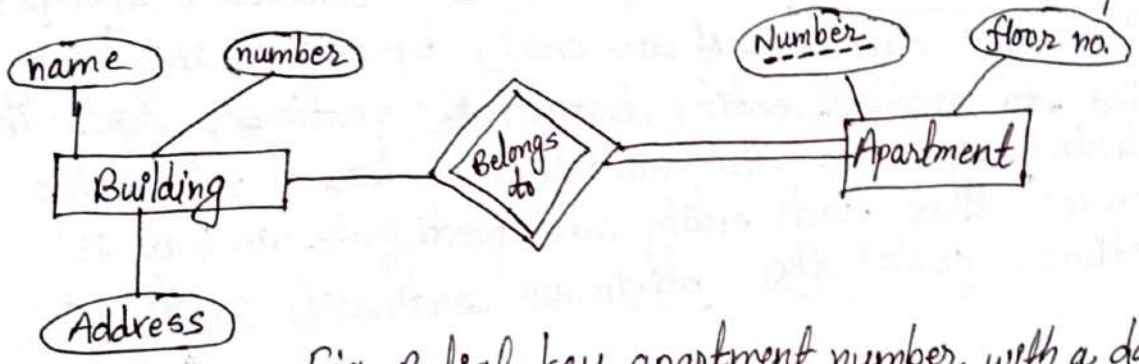


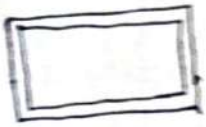
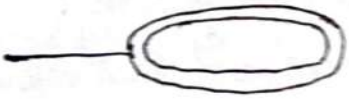
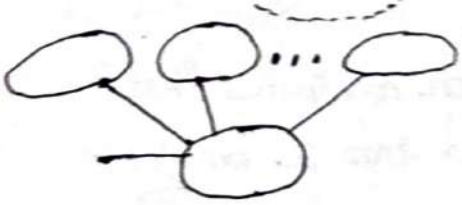
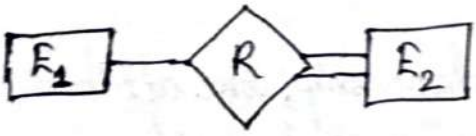
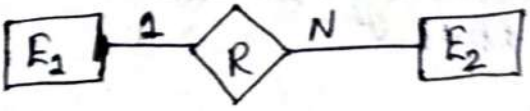
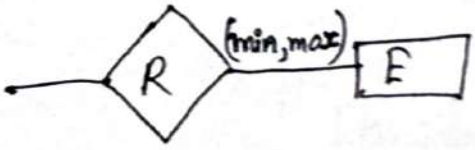
Fig. Partial key apartment number with a dashed line.

⊛ Drawing ER diagrams using E-R notations, naming conventions and design issues:

ER-diagram helps to explain the logical structure of database. It includes many specialized symbols, and it's meanings make this model unique. The purpose of ER diagram is to represent the entity framework ~~inf~~ infastructure. The components of ER-diagram are:-

→ Entities
→ Attributes
→ Relationships.

The following are the different symbols/notations used to draw ER diagrams: (Among these some are already written in first page of note but we will also ~~use~~ write here with some additional notations). These are the design choises for concephual design.

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key attribute
	Multivalued attribute
	Derived attribute.
	Composite attribute.
	Total Participation of E_2 in R
	Cardinality Ratio 1:N for $E_1:E_2$ in R
	Structural Constraint (min, max) on Participation of E in R.

Naming Conventions:

When designing a database schema, the choice of names for entity types, attributes, relationship types, and roles is not always straightforward. One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema. We choose to use singular names for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type. In ER diagrams we use the convention that entity type and relationship type names are uppercase letters, attribute names have their initial letter capitalized and role names are lowercase letters.

Design Issues:-

It is occasionally difficult to decide whether a particular concept in the miniworld should be modeled as an entity type, an attribute, or a relationship type. In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:

→ A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a refrence to another entity type.

→ It is often the case that a pair of such attributes that are inverses of one another are refined into a binary relationship.

→ Once an attribute is replaced by a relationship, the attribute itself should be removed from the entity type to avoid duplication and redundancy.

⊛. Concept of Enhanced ER (EER) Model:-

The model that consists all the modeling concepts of the ER-model and in addition includes/consists the concepts of subclass and superclass and the related concepts of specialization and generalization is called enhanced ER (i.e, EER) model. Also it includes concept of a category or union type, which is used to represent a collection of objects (entities). EER model is the improved or enhanced form of ER model to handle the complex applications better.

Features:

→ Creates a design more accurate to database schemas.
→ Reflects the data properties and constraints more precisely.
→ Includes all modeling concepts of ER model.
→ Includes concept of specialization and generalization.
→ Used to represent objects that is union of objects of different entity types.

Subclasses and Superclasses:

Superclass is an entity type that has a relationship with one or more subtypes. An entity cannot exist in database ~~just~~ only by being member of any super class. For example: Shape super class is having sub groups as Square, Circle and Triangle in the figure below.

Subclass is a group of entities with unique attributes. Subclass inherits properties and attributes from its super class. For example: Square, Circle and Triangle are the sub class of Shape super class.

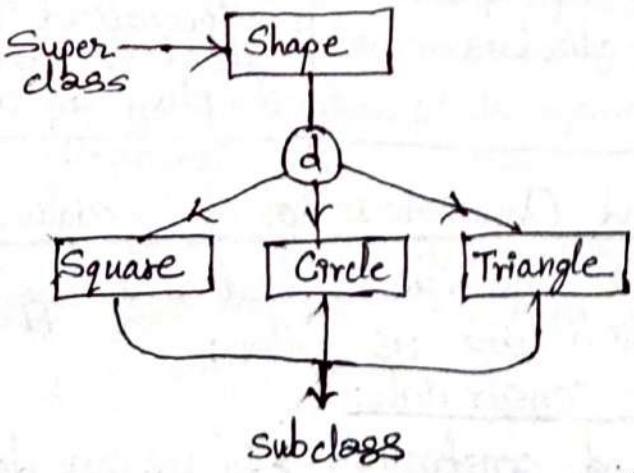


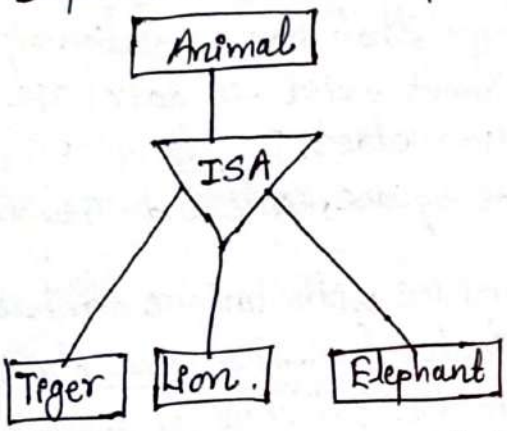
Fig: Subclass superclass relationship.

⊛. Concept of Specialization and Generalization:

Specialization → It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets. It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member. It defines one or more subclass for super class and also forms the super class/sub class relationship.

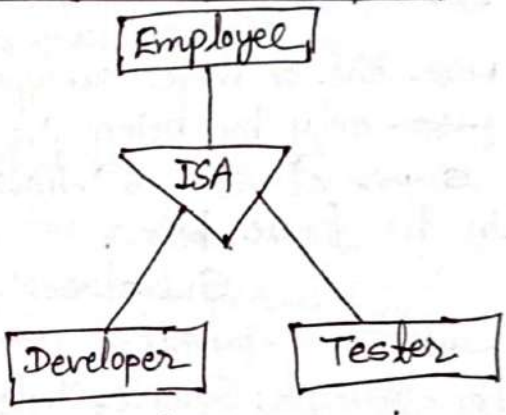
Generalization → Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity. In this concept sub-classes are combined to form a super-class.

Generalization Example



Here; Tiger, Lion and Elephant can all be generalized as animals.

Specialization Example



Here; Employee can be specialized as developer or tester base on what role they play in an organization.

⊛. Constraints and Characteristics of Specialization and Generalization:-

There are three constraints that may apply to specialization/generalization which are as follows:-

ⓐ. Membership constraints:

1) **Condition defined constraints** → If we can determine exactly those entities that ~~are~~ will become members of each subclass by a condition, the subclasses are called condition-defined classes.

Whenever any tuple is inserted into the database, it's membership in the various lower level entity-sets can be automatically decided by evaluating the respective membership predicates. Similarly when a tuple is updated, its membership in the various entity sets can be re-evaluated automatically.

ii) User defined constraints → If no condition determines membership, the subclass is called user-defined. Membership in the subclass is determined by the database users by applying an operation to add an entity to the subclass. Membership in the subclass is specified individually for each entity in the superclass for the user.

(b) Disjoint constraints:

i) Disjoint constraint → It specifies that the subclass of the specialization must be disjoint. Here an entity can be a member of at most one of the subclass of specialization and it is represented by d in EER diagram. If an entity can be a member of at most one of the subclass of the specialization, then the subclass are called disjoint.

ii) Overlapping constraint → It specifies that the subclasses are not constrained to be disjoint i.e, the same entity may be member of more than one subclass of the specialization and it is represented by o in EER diagram.

(c) Completeness constraints:

i) Total participation constraint → It specifies that every entity in the super class must be a member of some subclass in the specialization/generalization. It is represented by double line in EER diagram.

ii) Partial participation constraint → It allows every entity in the super class may not belong to a subclass and shown in EER diagram by a single line.

⊛. Differences between Primary key and foreign key:

Primary Key	Foreign Key
i) It helps us to uniquely identify a record in the table.	i) It is a field in the table that is the primary key of another table.
ii) Primary key never accepts null values.	ii) A foreign key may accept multiple NULL values.
iii) Primary key is a clustered index and data in the DBMS table are physically organized in the sequence of the clustered index.	iii) A foreign key cannot automatically create an index, clustered or non-clustered. However you can manually create an index on foreign key.
iv) We have single primary key in the table.	iv) We have multiple foreign keys in the table.

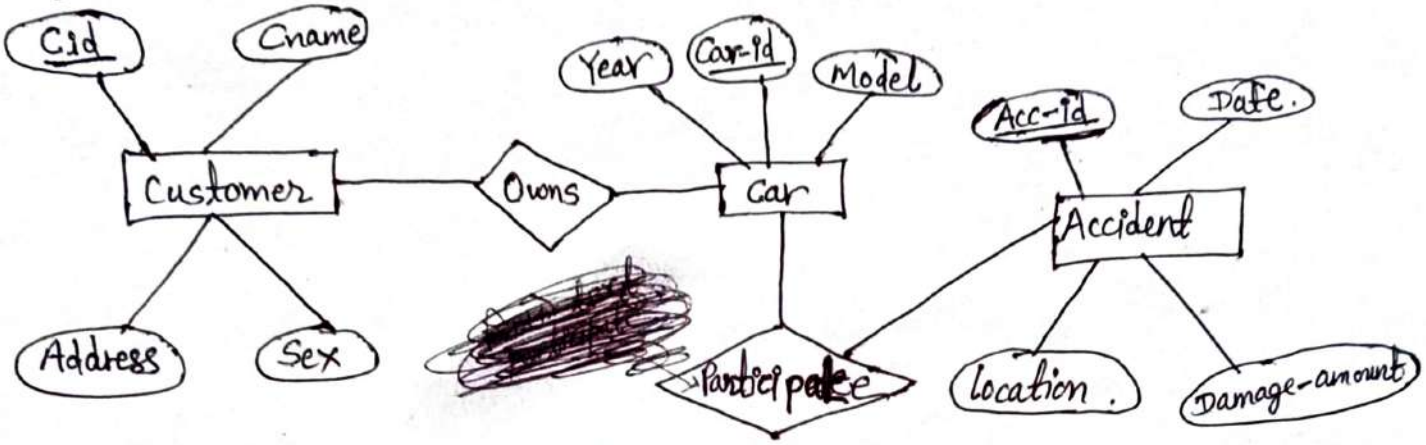
⊛. Differences between strong entity set and weak entity set:

Strong Entity Set.	Weak Entity Set.
i) Strong entity set always has a primary key.	i) It does not have enough attributes to build a primary key.
ii) It is represented by a rectangle symbol.	ii) It is represented by a double rectangle symbol.
iii) It contains a primary key represented by underline symbol.	iii) It contains a partial key which is represented by dashed underline symbol.
iv) The member of a strong entity set is called dominant entity set.	iv) The member of a weak entity set is called subordinate entity set.
v) Primary key is one of its attribute which helps to identify its member.	v) In a weak entity set, it is a combination of primary key and partial key of the strong entity set.
vi) In the ER diagram the relationship between two strong entity set is shown by using a diamond symbol.	vi) The relationship between one strong and a weak entity set is shown by using the double diamond symbol.

⊛. Examples of ER-diagram: [from kec book, page no. 59]

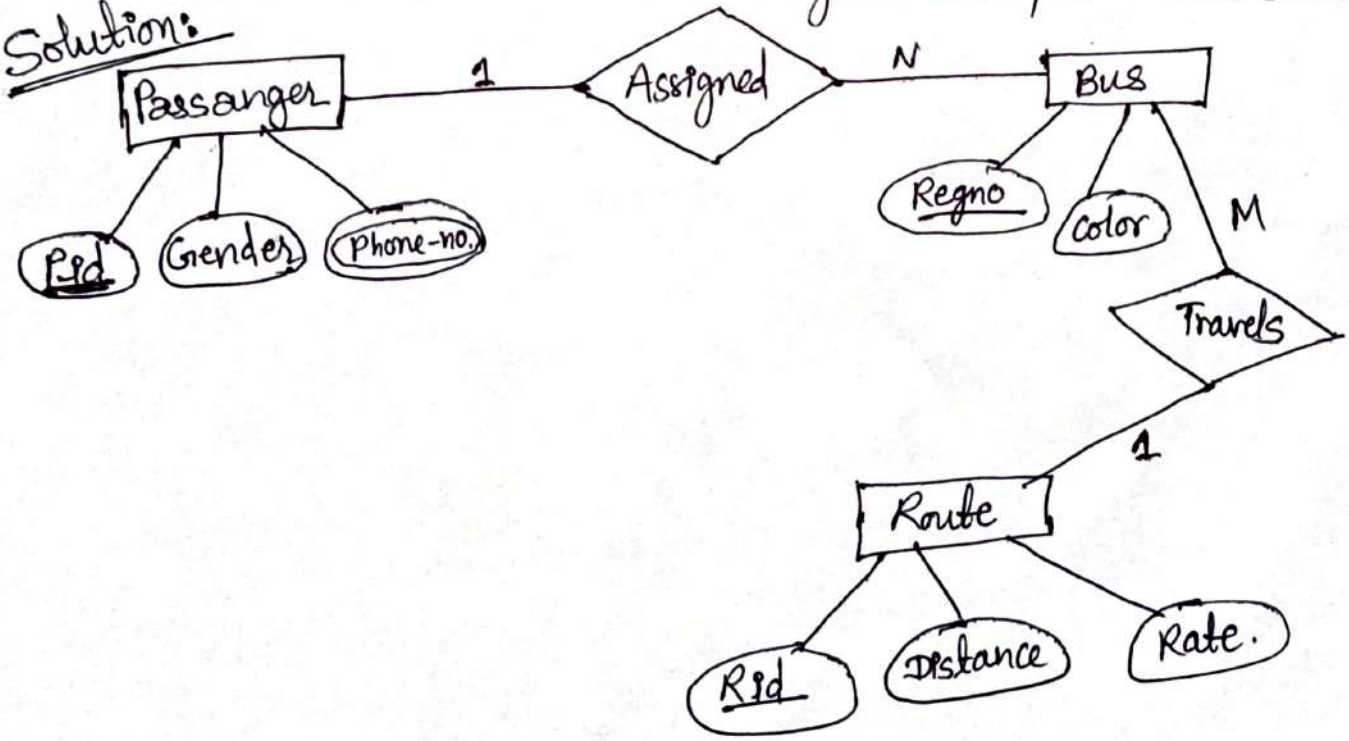
Example 1: Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Assume attributes of your own interest:

Solution:



Example 2: Consider a bus ticketing system that records information about the passanger, bus and route. Passanger is assigned to a bus travels to route. A bus contains many passangers and a passanger can be assigned into only one bus. Many buses travel in same route but a bus can travel in only one route. The attributes of passanger are ~~unique pass~~ pid (unique), gender and telephone (multi-valued). Similarly bus contains regno (unique) and color and route contains rid (unique), distance and rate (based on distance). Now draw E-R diagram to represent this situation.

Solution:



Unit-4

The Relational Data Model and Relational Database Constraints:

⊛ Relational Model Concepts:

Relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations.

Domain → A domain is a set of acceptable values that a column is allowed to contain. This is based on various properties and the data type for the column. It is the original set of atomic values used to model data. Atomic value mean that each value in the domain is indivisible as far as the relational model is concerned.

For example:- The domain of Marital status has a set of possibilities: Married, Single, Divorced.

Attribute → Attributes are the properties that define a relation.
For example:- ROLL_NO, NAME. etc.

Tuple → Each row in the relation is known as tuple. It contains the data records.

Relation → A relation in relational data model represents the respective attributes and the correlation among them.

Student (Relational Model).

Attributes (Columns)

Relation (Table)

Stu_id	Stu_name	Stu_address	Dept_id
12	Ramesh	Butwal	5
13	Hari	Kathmandu	3
14	Mahesh	Pokhara	1
15	Prem	Nepaljung	2
16	Sangita	Mahendranagar	4

Record or Tuple (Row)

Primary key attribute.

⊛. Characteristics of Relations:

i) Each relation in a database must have a distinct or unique name which would seperate it from the other relations in a database.

ii) A relation must not have two attributes with same name. Each attribute must have a distinct name.

iii) Duplicate tuples must not be present in relation.

iv) Each tuple must have exactly one data value for an attribute.

v) Tuples in a relation do not have to follow a significant order as the relation is not order-sensitive.

vi) Similarly, the attributes of a relation also do not have to follow a significant order.

⊛. Relational Model Notation:-

We will use following notation in our presentation:

i) A relation schema R of degree n is denoted by $R(A_1, A_2, \ldots, A_n)$.

ii) The uppercase letters Q, R, S denote relation names.

iii) The lowercase letters q, r, s denote relation states.

iv) The letters t, u, v denote tuples.

v) The name of relation schema such as STUDENT indicates current set of tuples in that relation whereas STUDENT(Name, Roll, ...) refers only to relation schema.

vi) An attribute A can be qualified with the relation name R to which it belongs by using the dot notation $R.A$ — for example: STUDENT.Name, STUDENT.Age etc.

vii) A n-tuple t in a relation $r(R)$ is denoted by $t = \langle v_1, v_2, \ldots, v_n \rangle$ where v_i is the value corresponding to attribute A_i.

→ Both $t[A_i]$ and $t.A_i$ (and sometimes $t[i]$ refer to the value v_i in t for attribute A_i.

→ Both $t[A_u, A_w, \ldots, A_z]$ and $t.(A_u, A_w, \ldots, A_z)$, where $A_u, A_w, \ldots, A_z$ is a list of attributes for R, refer to the subtuple of values $\langle v_u, v_w, \ldots, v_z \rangle$ from t corresponding to the attributes specified in the list.

Ⓐ. Categories of Constraints:

The types of constraints are as follows:-

i) Domain Constraints → Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A). The data types associated with domains typically include standard numeric data types for integers (such as short integer, integer, and long integer) and real numbers (float and double-precision float). Characters, Booleans, fixed-length strings are also available.

ii) Key Constraints → These are called uniqueness constraints since it ensures that every tuple in the relation should be unique. A relation can have multiple keys or candidate keys out of which we choose one of the key as primary key. We do not have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes. NULL values are not allowed in the primary key, hence Not NULL constraint is also a part of key constraint.

iii) Referential Integrity Constraints → The referential integrity constraints is specified between two relations or tables and used to maintain the consistency among the tuples in two relations. This constraint is compulsory through foreign key. When an attribute in the foreign key of relation R1 ~~is said~~ have the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to refrence or refer to the primary key of relation R2. The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some ~~tub~~ tuple in relation R2, or can take NULL values, but can't be empty.

⊛. Relational Database and relational database schemas:-

A relational database schema is a set of relation schemas $S = \{R_1, R_2, \ldots, R_m\}$ and a set of integrity constraints. A relational database state of S is a set of relation states $DB = \{r_1, r_2, \ldots, r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints.

In simple language, a relational database is an arrangement of relation states in such a manner that every relational database state fulfills the integrity constraints set on a relational database schema. When we refer to a relational database, we implicitly include both it's schema and it's current state. A database state that does not obey all the integrity constraints is ~~current state.~~ called not valid, and a state that satisfies all the constraints in the defined set of integrity constraints is called a valid state. In the context of relational database schema following points deserve a particular consideration:

i) A specific characteristic, that bears the same real-world concept may appear in more than one relationship with the same or a different name. For example; In Employees relation, Employee Id (EmpId) is represented in Vouchers as AuthBy and PrepBy.

ii) The specific real-world concept that appears more than once in a relationship should be represented by different names.

iii) The integrity constraints that are specified on database schema shall apply to every database state of that schema.

⊛. Entity Integrity, Refrential Integrity and Foreign Key:-

Entity Integrity → The entity integrity constraint states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in the relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example if two or more tuples had NULL for their primary keys, then we may not be able to distinguish them if we try to refrence them from other relations.

Refrential Integrity Constraint → The refrential integrity constraint is specified between two relations and is used to maintain the consistency among the tuples in the two relations. Refrential integrity constraints typically arise from the relationships among the entities represented by the relation schemas. This constraint is compulsory through foreign key. The values of foreign key in a tuple of relation R.1 is ~~said~~ can either take the values of the primary key for some tuple in relation R2, or can take NULL values, but can't be empty.

Foreign Key → It is the field in the table that is primary key of another table. A foreign key may accept multiple NULL values. A foreign key cannot automatically create an index, clustered or non-clustered. However we can manually create an index on foreign key. We have multiple foreign keys in the table.

⊛ Concept of insert, delete and update operations:

Q. Explain the operations and constraints violations?

Ans: Updates and retrieval are the two categories of operations on the relational model. The basic types of updates are:-

i) Insert → We use this operation in order to add new tuple in a relation. It is capable of violating any of the four constraints.

ii) Delete → We perform this operation in order to remove or delete a tuple in a relation. Under, this operation we can remove a particular data record from a table. It can only violate the referential integrity ~~constant.~~ constraint.

iii) Modify / Update → This operation causes a change in the values of some characteristic of existing tuples or accounting data tables.

Note: Retrival constraints do not cause a violation of integrity constraints.

⊛. Concept of transactions:-

A transaction is an executing program that includes some database operations, such as reading from database, or applying insertions, deletions, or updates to the database. A database application program running against a relational database typically executes one or more transactions. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.

A single transaction may involve any number of retrival operations. and any number of update operations. A large number of commercial applications running against relational databases in online transaction processing (OLTP) systems are executing transactions at rates that reach several hundred per second.

⊛. Advantages of using Relational Model:

→ A relational data model is simpler than the hierarchical and network model.

→ This model is concerned with data rather than structure. So this can improve the performance of model.

→ This model is easy to use since tables consisting of rows and columns is simple to understand.

→ This model is data independence since structure of database can be changed without having to change any application.

→ It makes possible for high-level query language like SQL to avoid complex database navigation.

⊛. Disadvantages of using Relational model:

→ Few relational databases have limits on field lengths which can't be exceeded.

→ Relational databases can sometimes become complex as the amount of data grows.

→ Complex relational database systems may lead to isolated databases where the information can't be shared from one system to another.

UNIT-5

The Relational Algebra and Relational Calculus:

Query Language → A query language is a language in which a user requests information from the database. It is of two types: procedural (in which user instructs system to perform sequence of operations on database. Example: Relational algebra) and non-procedural (in which user describes the desired information without giving specific procedure. Example: tuple relational calculus, SQL etc.).

⊛ Relational Algebra:

Concept only no need to remember all

It is a procedural query language which takes instances of relations as input and yeilds instances of relations as output. It uses operators to perform queries. An operator can be unary or binary.

Main operations of relational algebra

- **Fundamental Operations**
  - Select
  - Project
  - Union
  - Difference
  - Rename
  - Cartesian product
- **Additional Operations**
  - Set-insertion
  - Natural-join
  - Division
  - Assignment
- **Extended Relational Algebra Operations**
  - Generalized projection
  - Aggregate function
  - Outer join
  - Null values
- **Unary Relational Operations**
  - Select (symbol: σ)
  - Project (symbol: π)
  - Rename (symbol: ρ)

⊛ Unary Relational Operations: SELECT and PROJECT

The relational algebra operations that uses single relation (table) are called unary relational operations.

ⓐ Select (σ): The select operation is used for selecting a subset of the tuples according to a given selection condition. It is denoted by Sigma (σ) symbol.

Syntax: σ <selection condition> (R)

where, R stands for relation which is the name of the table.

Comparision operators (<, >, ≤, ≥, =, ≠) can be used to specify conditions required for selection tuples from a relation. Furthermore logical operations AND (∧), OR (∨) and NOT (−) are used to combine two or more conditions.

Example: Let's take a student relation.

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	22
11	Abin	Ktm	2	17
12	Aarav	Ktm	1	21
13	Ashna	Palpa	3	45
14	Anuj	Pokhara	4	23

i) Find records of all students of address 'Palpa'.

$$\sigma_{Stu_address = \text{"Palpa"}}(Student)$$

ii) Find all students of age greater than 20 or of address 'Ktm'.

$$\sigma_{Stu_address = \text{"Ktm"} \vee Age > 20}(Student)$$

(b) Projection (π): The project operation is used to select certain columns from the table and discards the other columns. It is denoted by Pie (π) symbol.

Syntax: $\pi_{<attribute-list>}(R)$

Example: We are using above student table,

i) Display name and id of the student.

$$\pi_{Stu_id, Stu_name}(Student)$$

ii) Display name and age of students

$$\pi_{Stu_name, Age}(Student)$$

(*) Combining Selection and Projection Operations:

The selection and projection operators are combined to perform projection with selection operation.

Syntax: $\pi_{<attribute-list>}(\sigma_{<selection\ condition>}(R))$

Example: We are using above student table,

i) Find name, address, age of student whose age less than or equal to 25.

$$\pi_{Stu_name, Stu_address}(\sigma_{age \leq 25}(Student))$$

ii) Find name of students whose age is greater than 20 and of address "Palpa".

$$\pi_{Stu_name}(\sigma_{age > 25 \wedge Stu_address \text{"Palpa"}}(Student))$$

⊛. Sequence of Operations and the RENAME Operation:

We can either write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. We must give names to the relations that hold the intermediate results. For example: To retrive, first name, last name and salery of all employees who work in department number 5, we must apply SELECT and PROJECT operation as follows:-

$$\pi_{Fname, Lname, Salery}(\sigma_{Dno=5}(EMPLOYEE)).$$

This is in-line relational algebra expression, also known as in-line expression.

Alternatively, we can show the sequence of operations, giving a name to each intermediate relation, and using the assignment operation, denoted by ← (left arrow) as follows:-

$$DEP5_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$

$$RESULT \leftarrow \pi_{Fname, Lname, Salery}(DEP5_EMPS).$$

It is sometimes simpler to break down a complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression. This can be useful with more complex operations such as UNION and JOIN.

RENAME Operation:- We can also define a formal RENAME operation which can rename either the relation name or attribute names, or both as a unary operator. The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:

$$\rho_{S(B1,B2,\ldots,Bn)}(R) \quad \underline{or} \quad \rho_S(R) \quad \underline{or} \quad \rho_{(B1,B2,\ldots,Bn)}(R).$$

where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and B1, B2, ... Bn are new attribute names.

⊛. Relational Algebra Operations from Set Theory:-

i) **Union Operation (U):-** Let two union-compatible relations be R and S. Then, the union operation (U) ~~is den~~ between R and S is denoted by RUS, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

Example: Let's take following two tables namely morning shift Employee as "Memployee" and Day shift employee as "Demployee".

Memployee

eId	name	salery
e1	Rajan	34,000
e2	Aarab	45,000
e3	Abm	55,000
e4	Ashna	24,000

Demployee

eId	name	salery
e1	Rajan	34,000
e5	Umesh	78,000
e8	Anisha	55,000
e4	Ashna	33,000

Now Memployee U Demployee is as follows:-

eId	name	salery
e1	Rajan	34,000
e2	Aarab	45,000
e3	Abm	55,000
e4	Ashna	24,000
e5	Umesh	78,000
e8	Anisha	55,000
e4	Ashna	33,000

Since e1 Rajan 34,000 of Demployee is repeating (i.e, duplicate data) so eliminated

ii) **Intersection Operation (∩):-** It is denoted by symbol ∩ and it returns a relation that contains tuples that are in both of its argument relations.

For example:- From above tables Memployee and Demployee Memployee ∩ Demployee is as follows:-

eId	name	Salery
e1	Rajan	34,000

iii) **Difference (−):-** It is denoted by minus sign (−). It finds tuples that are in one relation, but not in another. Thus results in relation containing tuples that are in R but not in S.

For example: Memployee − Demployee is:-

eId	name	salery
e2	Aarav	45000
e3	Abm	55000
e4	Ashna	24000

iv) Cartesian Product (x) :- This type of operation is helpful to merge columns from two relations. The cartesian product operation does not require relations to union-compatible i.e, the involved relations may have different schemas. The cartesian product of two relations R and S is denoted by R x S, is the set of all possible combinations of ~~tub~~ tuples of two relations.

For Example :

Department

Dept_id	Dept_name	Dept_block no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400

Staff

Staff_id	staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1

Now Department x Staff is as follows:-

D.Dept_id	Dept_name	Dept_block no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	22	Pratima	2
1	Computer	100	33	Madan	1
2	Mathematics	200	11	Mohan	1
2	Mathematics	200	22	Pratima	2
2	Mathematics	200	33	Madan	1
3	Economics	300	11	Mohan	1
3	Economics	300	22	Pratima	2
3	Economics	300	33	Madan	1
4	Account	400	11	Mohan	2
4	Account	400	22	Pratima	1
4	Account	400	33	Madan	2

⊛. Binary Relation Operations : JOIN and DIVISION:-

The operations that are used to perform operations into multiple tables are called binary relation operations.

ⓐ Join operation: Join operation is essentially a cartesian product followed by a selection criteria. It is denoted by ⋈.

Types of Join operations

- Inner Join
  - Theta join
  - Equi join
  - Natural join
- Outer Join
  - Left outer join
  - Right outer join
  - Full outer join

1) **Inner Join:** In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded.

i) **Theta join** → The general case of join operation is called a theta join. It is denoted by symbol θ. The theta condition consists one of the comparision operators (=, <, <=, >, >=, <>).

Syntax: $A \bowtie_{\theta} B$ where, A and B are any two relations and θ is a join condition.

Example:

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1

Now Department $\bowtie_{D.Dept_id > S.Dept_id}$ (Staff).

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
2	~~Computer~~ Mathematics	200	11	Mohan	1
2	~~Computer~~ Mathematics	200	33	Madan	1
3	Economics	300	11	Mohan	1
3	Economics	300	22	Pratima	2
3	Economics	300	33	Madan	1

ii) **Equi Join** → When join condition is '=' i.e, θ is =, the operation is called equijoin.

Syntax: $A \bowtie_{=} B$ where, A and B are any two relations and = is a join operation.

Example:- Let we take above two relations Department and Staff.

Department ⋈ D.Dept_id = S.Dept_id (Staff).

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2

iii) Natural Join → Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same. It allows us to combine certain selections and a cartesian product into one operation. It is denoted by join symbol, ⋈. The natural ~~joins~~ join performs a join by equating the attributes with the same name and then eliminates duplicate attributes.

Example: Let we take above two relations Department and Staff.

Department ⋈ Staff

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name.
1	Computer	100	11	Mohan
1	Computer	100	33	Madan
2	Mathematics	200	22	Pratima

Same id भांको लेखेर आकों id eliminate गरेको

2) Outer Join: In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria. Thus the outer join is an extension of the join operation to deal with missing information.

i) Left Outer Join (⟕) → It allows keeping all tuple in left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with NULL values.

Example: Department ⟕ Staff.

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	NULL	NULL	NULL

ii) Right Outer Join (⟖) → This operation allows keeping all tuple in the right relation. However, if there is no matching tuple found in the left relation, then the attributes of the left relation in the join result are filled with NULL values.

Example:- Let we have following two relations:-

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
3	Economics	300

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	3

Now Department ⟖ Staff is as follows:-

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
~~1~~	~~Computer~~	~~100~~	~~33~~	~~Madan~~	~~1~~
NULL	NULL	NULL	22	Pratima	2
3	Economics	300	33	Madan	3

iii) Full Outer Join (⟗) → It includes all tuples in left hand relation and from the right hand relation. In a full outer join, all ~~tub~~ tuples from both relations are included in the result, irrespective of matching condition.

Example:- Let we have following two relations:

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
3	Economics	300

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2

Now Department ⟗ Staff is as follows:-

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
3	Economics	300	NULL	NULL	NULL
NULL	NULL	NULL	22	Pratima	2

⊛. Division operation (÷) :- It is denoted by symbol ÷ and is suited to queries that include the phrase "for all". It takes two relations and builds another relation, consisting of values of an attribute of one relation that match all the values in the other relation.

Examples:-

sno	pno
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

A

pno
P2

B1

pno
P2
P4

B2

pno
P1
P2
P4

B3

sno
S1
S2
S3
S4

A/B1

sno
S1
S4

A/B2

sno
S1

A/B3

⊛ Additional Relational Operations:

⊛. Concept of Generalized Projection:- Generalized projection is enhanced version of project operation. It allows us to write aromatic operations containing attribute names and constants in projection list. General form of generalized projection is as follows:

$$\Pi_{F1, F2, F3, \ldots, Fn}(E).$$

where, E is a relational algebra expression and F_i $(i=1, 2, \ldots n)$ is an attribute or arithmetic expression containing attributes and constants.

Example:- Let's take an employee relation as follows:-

Employee

eid	name	Age	Salery	Address
e1	Rajan	33	34000	Ktm
e2	Arav	17	45000	Pokhara
e3	Abin	22	55000	Palpa
e4	Ashna	19	24000	Ktm

i) Find name and ~~salery~~ salery of all employees by increasing their salery by 15%.

$$\Pi_{name, salery = salery + salery * 0.15}(Employee)$$

ii) Increase the salery of all employees whose age greater than 20 by 5%.

$$\Pi_{eid, name, age, salery = salery + salery * 0.05}(\sigma_{age > 20}(Employee))$$

Aggregate Functions:

Aggregate functions are algebraic functions that take a collection of values as input and return a single value as a result. It is denoted by symbol $\mathcal{G}$ read as "calligraphic G". General form of aggregate operation in relational algebra is;

$$G1, G2 \ldots Gn\ \mathcal{G}_{F1(A1), F2(A2), \ldots, Fn(An)}(E).$$

where, E is any relational-algebra expression.
G1, G2, ... Gn is a list of attributes on which to group and it can be empty.
Each F_i is an aggregate function and each A_i is an attribute name.

There are five aggregate functions:

- → SUM: sum of values
- → AVG: average value
- → MIN: minimum value
- → MAX: maximum value
- → COUNT: number of values.

Example:- Consider Employee relation that we have in example of generalized projection before;

i) Find total number of employees.

$$\mathcal{G}_{COUNT(eid)}(Employee)$$

ii) Find average age of employees of address 'ktm'

$$\mathcal{G}_{AVG(Age)}(\sigma_{address = "ktm"}(Employee))$$

iii) Find minimum and maximum age of the employee.

$$\mathcal{G}_{MIN(Age), MAX(Age)}(Employee).$$

iv) Find average salery of employee in each address level.

$$_{Address}\mathcal{G}_{AVG(Salery)}(Employee).$$

v) Find total salery of employees.

$$\mathcal{G}_{SUM(Salery)}(Employees).$$

⊛ Tuple Relational Calculus:

Tuple Relational Calculus is a non-procedural query language unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do. In Tuple calculus, a query is expressed as;

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

where, t = resulting tuples.
$P(t)$ = known as predicate and these are the conditions that are used to fetch t.

Thus, it generates set of all tuples t, such that Predicate $P(t)$ is true for it. $P(t)$ may have various conditions logically combined with OR ($\vee$), AND ($\wedge$), NOT ($-$). It also uses quantifiers $\exists$ (there exists) and $\forall$ (for all).

Example:- Consider a Loan relation as follows:-

Loan number	Branch name	Amount
L33	ABC	10,000
L35	DEF	15,000
L49	GHI	9000
L98	DEF	65000

Q) Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

$$\{t \mid t \in loan \wedge t[amount] >= 10000\}$$

⊛. Domain Relational Calculus:

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In domain relational calculus, a query is expressed as;

$$\{\langle x_1, x_2, x_3, \ldots, x_n \rangle \mid P(x_1, x_2, x_3, \ldots, x_n)\}$$

where, $\langle x_1, x_2, x_3, \ldots, x_n \rangle$ represents resulting domain variables and $P(x_1, x_2, x_3, \ldots, x_n)$ represents the condition or formula equivalent to Predicate calculus.

Predicate Calculus formula:

→ Set of all comparision operators
→ Set of connectives like and, or, not.
→ Set of quantifiers.

Example: Consider the following relations Loan ~~and Borrower~~.

Loan

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150
L10	Sub	90
L08	Main	60

Borrower

Customer name	Loan number
Ritu	L01
Debomit	L08
Soumya	L03

i) Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

$$\{\langle l, b, a\rangle \mid \langle l, b, a\rangle \in loan \wedge (a \geq 100)\}$$

Result:

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150

⊛. Differences between Relational Algebra & Relational Calculus:

Relational algebra	Relational Calculus
i) Relational algebra is a procedural language.	i) Relational calculus is a declarative language.
ii) It states how to obtain the result.	ii) It states what result we have to obtain.
iii) It describes the order in which operations have to be performed.	iii) It does not specify the order of operations.
iv) It is not domain dependent.	iv) It can be domain dependent.
v) It is close to programming language.	v) It is close to natural language.

Note:- In addition have a look at relational algebra examples page no. 85 of kec book.

Unit-6
(SQL) Structured Query Language

Structured Query Language (SQL) was the first commercial language for relational model of database. It is a database query language, used for storing and managing data in Relational DBMS. SQL uses the terms **table, row** and **column** for the formal relational model terms relation, tuple and attribute respectively. The main SQL command for data definition is the **CREATE** statement, which can be used to create schemas, tables etc. Today almost all RDMS (MySQL, Oracle, Ms.Access etc.) use SQL as the standard database query language.

⊛ CREATE TABLE Command in SQL:

CREATE TABLE Command tells the database system to create a new table. This command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints. The attributes are specified first and each attribute is given a name, a data type to specify it's domain of values and possibly attribute constraints, such as NOT NULL. The basic syntax of CREATE TABLE statement is as follows:-

```
CREATE TABLE table_name(
        column 1  datatype,
        column 2  datatype,
        :
        column N  datatype,
        PRIMARY KEY (one or more columns)
);
```

In brackets comes the list defining each column in the table and what sort of data type it is. The relations declared through CREATE TABLE command are called base tables (or base relations); this means that the table and its rows are actually created and stored as a file by DBMS.

Example: Creating table for Employee.

```
CREATE TABLE EMPLOYEE(
        Fname     VARCHAR(15)   NOT NULL,
        Lname     VARCHAR(15)   NOT NULL,
        Ssn       CHAR(9),
        Bdate     DATE,
        Sex       CHAR,
        Salery    DECIMAL(10,2),
        Dno       INT           NOT NULL,
        PRIMARY KEY (Ssn)
);
```

⊛. Attribute Data Types and Domains in SQL:-

The basic data types available for attributes include numeric, character string, bit string, Boolean, date, and time.

Numeric → Numeric data types include integer numbers of various sizes (INTEGER, INT, and SMALLINT) and floating point numbers of various precision (FLOAT and DOUBLE PRECISION). Formatted numbers can be declared by using DECIMAL(i,j) where i is the total no. of decimal digits and j is the no. of digits after decimal point.

Character-string → Character-string data types are either fixed length — CHAR(n) or CHARACTER(n), where n is the number of characters or varying length — VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters.

Bit-string → Bit-string data types are either of fixed length n-BIT(n) or varying length — BIT VARYING(n), where n is the maximum number of bits. The default for n, the length of a character string or bit string is 1. Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings; for example, B'10101'.

Boolean → A Boolean data type has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, third possible value for a Boolean data type is UNKNOWN.

Date → The DATE data type has ten positions, and it's components are YEAR, MONTH and DAY in the form YYYY-MM-DD. The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form of HH:MM:SS. Only valid dates and times should be allowed by the SQL implementation.

Domains in SQL → A domain can be declared and the domain name can be used with attribute specification. This makes it easier to change the data type for a domain that is used by numerous attributes in a schema, and improves schema readability. For example we can create a domain SSN_TYPE by the following statement;

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);
```

We can use SSN_TYPE in place of CHAR(9). A domain can also have an optional default specification via a DEFAULT clause. Notice that domains may not be available in some implementations of SQL.

⊗. ALTER and DROP commands →

ALTER command → The alter command is used to modify an existing database, table, view or other database objects that might need to change during the life cycle of a database. Let's suppose that we have completed our database design and it has been implemented. Our database users are using it and then they realize some of the vital information was left out in the design phase. They don't want to lose the existing data but just want to incorporate the new information. The alter command comes in handy in such situations. We can use the alter command to change the data type of a field say string to numeric, change the field name to new name or even add a new column in a table.

Syntax:

```
ALTER TABLE 'table_name' ADD COLUMN 'column_name'
                                    'data_type';
```

DROP command → The DROP command is used to delete a database from server or to delete an object (like Table, Column) from a database.

Syntax:

To drop table: DROP TABLE `table_name`;

Specifying Constraints:

Constraints: Constraints are the rules that we can apply on the type of data in a table. The available constraints in SQL are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT. We can specify constraints at the time of creating the table as below syntax.

```
CREATE TABLE table_name(
        column1 data_type(size) constraint_name,
        column2 data_type(size) constraint_name,
        :
        );
```

Specifying Attribute Constraints and Attribute Defaults:

Because SQL allows NULLs as attribute values, a constraint NOT NULL may be specified if NULL is not permitted for a particular attribute. This is always implicitly specified for any other attributes whose values are required not to be NULL.

It is also possible to define a default value for an attribute by appending the clause DEFAULT <value> to an attribute definition. The default value is included in new tuple if an explicit value is not provided for that attribute. If no default clause is specified, the default value NULL for attributes that do not have the NOT NULL constraint.

Another type of constraint can restrict attribute or domain values using the CHECK clause. For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then we can change the attribute declaration of Dnumber in the DEPARTMENT table to the following:

Dnumber INT **NOT NULL CHECK** (Dnumber > 0 **AND** Dnumber < 21);

⊛. Specifying Key and Referential Integrity Constraints:-

Because keys and referential integrity constraints are very important, there are some special clauses within the CREATE TABLE statement to specify them. The **PRIMARY KEY** clause specifies one or more attributes that makes up the primary key of a relation. For example, the primary key of DEPARTMENT can be specified as;

Dnumber INT **PRIMARY KEY**

The **UNIQUE** clause specifies alternate (unique) keys, also known as candidate keys. The **UNIQUE** clause can also be specified directly for a unique key if it is single attribute as;

Dname VARCHAR (15) **UNIQUE**

Referential integrity is specified via the **FOREIGN KEY** clause. A referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is updated.

Basic Retrival Queries:

⊛. SELECT-FROM-WHERE Structure:-

The basic form of the select-from-where block is formed of the three clauses SELECT, FROM and WHERE and has the following form:

SELECT <attribute list>
FROM <table list>
WHERE <condition> ;

where,

<attribute list> is a list of attribute names whose values are to be retrived by the query.

<table list> is a list of the relation names required to process the query.

<condition> is a conditional (Boolean) expression that identifies the tuples to be retrived by the query.

Example1: Retrive the birth date and address of the employee whose name is 'John B. Smith'.

Solution:

```
SELECT   Bdate, Address
FROM     EMPLOYEE
WHERE    Fname='John' AND Minit='B' AND Lname='Smith';
```

⇒ This query in the example1 involves only the EMPLOYEE relation listed in the FROM clause. The query selects individual EMPLOYEE tuples that satisfy the condition of WHERE clause, then projects the result on the Bdate and Address attributes listed in the SELECT clause.

Example 2: Retrive the name and address of all employees who work for the 'Research' department.

Solution:

```
SELECT   Fname, Lname, Address
FROM     EMPLOYEE, DEPARTMENT
WHERE    Dname='Research' AND Dnumber=Dno;
```

⊛. Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

Ambiguous Attribute Names: In SQL, the same name can be used for two (or more) attributes as long as the attributes are in different tables. If this is the case, and a multitable query refers to two or more attributes with the same name then we must **qualify** the attribute name with the relation name to prevent ambiguity. This is done by prefixing the relation name to the attribute name and separating the two by a period.

Example :-

```
SELECT   EMPLOYEE.Fname, Address
FROM     EMPLOYEE, DEPARTMENT
WHERE    Dname='Research' AND Dnumber=Dno;
```

Here, writing EMPLOYEE.Fname we are qualifying attribute name Fname with relation name EMPLOYEE (i.e, we are prefixing relation name EMPLOYEE to attribute name Fname) to specify which one we are referring to either EMPLOYEE or DEPARTMENT. EMPLOYEE.Fname shows we are refering to EMPLOYEE. This helps to prevent from ambiguity.

Aliasing and Tuple Variables:

We can rename the table names to shorter names by creating an alias for each table name to avoid repeated typing of long table names, which is called aliasing. An alias follow the keyword AS, as shown below in the example.

Example :- For each employee, retrive the employee's first and last name and the first and last name of his or her intermediate supervisor.

Solution:

```
SELECT   E.Fname, E.Lname, S.Fname, S.Lname
FROM     EMPLOYEE AS E, EMPLOYEE AS S
WHERE    E.Super_ssn=S.Ssn;
```

⇒ In this case we declare alternative relation names E and S, called aliases or tuple variables, for the EMPLOYEE relation.

Renaming: It is also possible to rename the relation attributes within the query in SQL by giving them aliases. For example, if we write

EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Dno)

in the form of clause, then Fn becomes alias for Fname, Mi for Minit, Ln for Lname and so on. We can use this alias-naming or renamming mechanism in any SQL query to specify tuple variables for every table in the WHERE clause. This practice is recommended since it results in queries that are eaiser to comprehened.

⊛. Unspecified WHERE Clause and Use of the Asterisk:

Unspecified or a missing WHERE clause indicates no condition on tuple selection. Hence all tuples of the relation specified in the FROM clause qualify and are selected for the query result. If more than one relation is specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT – all possible tuple combinations – of these relations is selected. If any condition is incorrect or unspecified then very large relations may result.

Example:- Select all ~~EMPLOYEE Ssns and all~~ combinations of EMPLOYEE Ssn and DEPARTMENT Dname in the database.

Solution:

```
SELECT  Ssn, Dname
FROM    EMPLOYEE, DEPARTMENT;
```

Use of Asterisk: To retrive all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL; we just specify an asterisk (*), which stands for all the attributes. This * can also be prefixed by the relation name or alias. For Example EMPLOYEE* refers to all attributes of the EMPLOYEE table.

⊛ Pattern matching and arithmetic operators:

Substring Pattern Matching:- This feature allows comparision conditions on only parts of a character string, using the LIKE comparision operator. This can be used for string pattern matching. Partial strings are specified using two reserved characters: % replaces an arbitrary number of zero or more characters, and the underscore (_) replaces a single character.

Patterns are case sensitive. Special characters (percent, underscore) can be included in patterns using an escape character '\' (backslash).

Examples:

i) 'Raj%' matches any string starting with "Raj".

ii) '%Raj' matches any string ending with "Raj".

iii) '__91' matches with any string ending with "91", with any two characters before that.

iv) '____' matches any string with exactly four characters.

Using 'LIKE' operator example: (also part of pattern matching topic)

Obtain roll numbers and names of all students whose names end with 'Mohan'.

```
SELECT  rollNo, name
FROM    STUDENT
WHERE   name LIKE '%Mohan';
```

Note: % & (_) are called wildcard operators

Arithmetic Operators:-

Operator	Operation	Description
+	Addition	Add values on either side of operator.
−	Subtraction	Used to subtract the right hand side value from left hand side value.
*	Multiplication	Multiplies the values present on each side of the operator.
/	Division	Divides left hand side value from the right hand side value.
%	Modulus	Returns the remainder.

Comparision Operators:- =, >, <, <=, >=, <> or != , !>, !< etc. are comparision operators used in SQL.

(also called WHERE clause operators)

(not greater than)

Logical Operators:- AND, OR, NOT and following are used in SQL;

BETWEEN → Searches the values within the range mentioned.

EXISTS → Used to search for the row's presence in table.

LIKE → Compares a pattern using wildcard operators.

ALL → Used to compare specific value to all other values in set.

ANY → Compares a specific value to any of the values in set.

Example: Retrive all employees in department 5 whose salery is between $30,000 and $40,000.

```
SELECT *
FROM   EMPLOYEE
WHERE  (Salary BETWEEN 30000 AND 40000) AND Dno=5;
```

(specifies all)

Comparisons involving NULL:

SQL represents "this has no value" by the special non-value NULL. NULL isn't a value it's just a representation that there is no any value. The result of comparing anything to NULL, even itself, is always NULL. A comparision to NULL is never true or false. Since NULL can never be equal to any value, it can never be unequal, either.

The correct way to write the queries is instead of using boolean comparision operators such as less-than and greater-than, equal-to and not-equal-to, these queries must be written with the special comparision operator **IS NULL**. The IS NULL operator tests whether a value is null or not null, and returns a boolean.

Example:- select * from table where column is not null;

we can write in single line as this or multiple lines as we used before.

Nested Queries:-

If a query is written inside a query, then it is called nested query. In nested queries the result of inner query is used in the execution of outer query. It is embedded within the WHERE clause, enclosed within parenthesis. There are mainly two types of nested queries:

1) Independent nested queries → The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc. are used in writing independent nested queries.

ii) Co-related nested queries → The execution of inner query is dependent of outer query and the result of inner query is used in the execution of outer query. Various operators like AND are used in co-related nested queries.

For Example:

```
SELECT *
FROM CUSTOMERS
WHERE ID IN(
        SELECT ID
        FROM CUSTOMERS
        WHERE Salary > 4500);
```

⊛ Concept and example of INSERT, DELETE and UPDATE commands:

Insert Command → INSERT is used to add a single tuple (row) to a relation (table). We must specify the relation name and a list of values for the tuple.

Example:

```
INSERT INTO EMPLOYEE
VALUES ('Roshan', 'Bist', '2000-7-5', '50000', 4);
```

A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command. This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.

Example:

```
INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn)
VALUES ('Roshan', 'Bist', 4, '653298653');
```

DELETE Command → The DELETE command removes tuples from a relation. It includes a WHERE clause to select the tuples to be deleted. Tuples are explicitly deleted from only one table at a time. Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command. A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database

as an empty table. We must use the DROP TABLE command to remove the table definition.

Example:

```
DELETE FROM EMPLOYEE
WHERE Lname='Bist';
```

UPDATE Command → The UPDATE command is used to modify attribute values of one or more selected tuples. As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation. An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values.

For Example:- To change the location and controlling department number of project number 10 to 'Butwal' and 5, respectively we write the query as following;

```
UPDATE PROJECT
SET    Plocation='Butwal', Dnum=5
WHERE  Pnumber=10;
```

Several tuples can be modified with a single UPDATE command. Example: Employees in the 'Research' department a 10% raise in salary, the query is as follows:

```
UPDATE EMPLOYEE
SET    Salary = Salary * 1.1
WHERE  Dno = 5;
```

It is possible to specify NULL or DEFAULT as the new attribute value.

⊛. Concept of views:

A view in SQL terminology is a single table that is derived from other tables. These other tables can be base tables or previously defined views. A view does not necessarily exist in physical form; it is considered to be a virtual table, in contrast to base tables, whose tuples are always physically stored in the database. This limits the possible update operations that can be applied to views, but it does not provide any limitations on ~~quering~~ querying a view.

⊛. CREATE VIEW Command:

In SQL, the command to specify a view is CREATE VIEW. A view can be created from a single table or multiple tables. The view is given a table name, a list of attribute names and a query to specify the contents of the view.

Syntax:

```
CREATE VIEW view_name AS
SELECT  column1, column2, ...
FROM    table_name
WHERE   condition;
```

Example: Create a View named DetailsView from the table Student Details, whose student ID is smaller than 5.

Query:

```
CREATE VIEW DetailsView AS
SELECT  Name, Address
FROM    Student Details
WHERE   S_ID < 5;
```

UNIT-7

Relational Database Design

⊛. Relational Database Design Using ER-to-Relational Mapping:

After designing the ER diagram of system, we need to convert it to Relational models which can directly be implemented by any RDBMS like Oracle, MySQL etc. To reduce given ER diagram into tables normally we divide ER diagram into following sections:

1. Mapping Strong entity sets to ER
2. Mapping Weak entity sets to ER
3. Mapping Relation sets to ER
   - i) Mapping of Binary 1:1 relation types to ER
   - ii) Mapping of Binary 1:N relationship types to ER
   - iii) Mapping of Binary M:N relationship types to ER.
4. Mapping of multivalue attributes to ER
5. Mapping composite attributes to ER
6. Mapping of N-ary relationship types to ER
7. Mapping specialization/generalization to ER
8. Mapping Aggregation to ER.

1. Mapping Strong entity sets to ER:

→ Create table for each of the strong entity.
→ Entity's attributes should become fields of table with their respective data types.
→ Declare key attribute of strong entity set as primary key of the table.

Example: Let's take a strong entity set Employee with following attributes;

Here, we simply create a table "Employee" with fields Eid, Name, Salary and Address and make Eid as primary key of the created table as below;

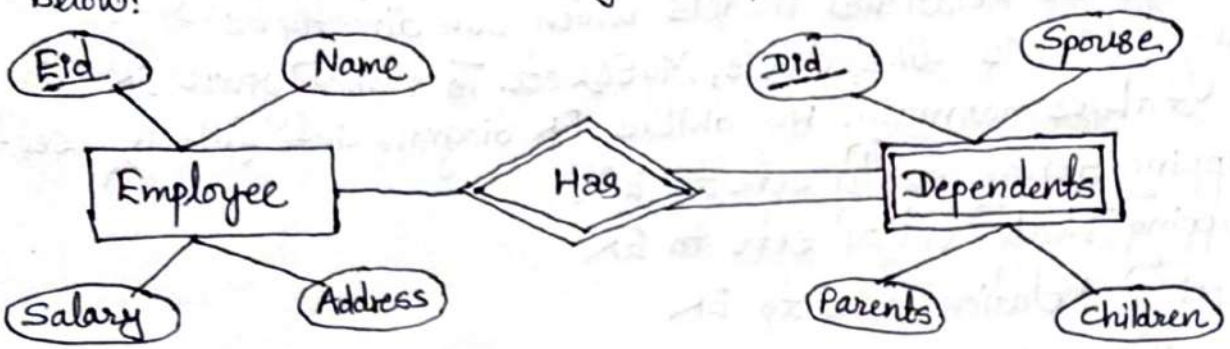
Employee

Eid	Name	Salary	Address

2. Mapping Weak entity sets to ER:

→ Create table for weak entity set.
→ Add all it's attributes to table as field.
→ Add the primary key of identifying entity set.
→ Declare all foreign key constraints.

Example: Let's take weak entity set Dependents as shown in ER diagram below:



Here, to draw table of weak entity set 'Dependents' we simply set all their attributes and also set primary key of Employee to the Dependents table as below:

Employee

Eid	Name	Salary	Address

Dependents

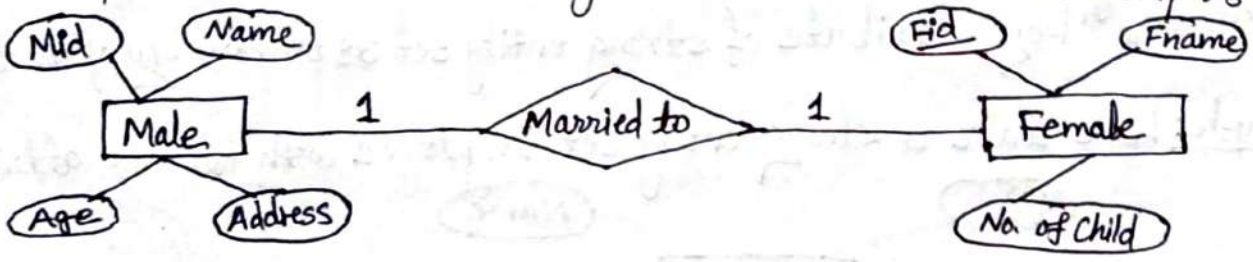
Did	Did	Parents	Spouse	Children

3. Mapping Relation sets to ER

i) Mapping of Binary 1:1 relation types to ER:

For constructing table from a binary one to one relationship we set primary key of any one of the entity set as foreign key.

Example: Let's take ER diagram with one to one relationship as below:



Here we can set Mid as foreign key to Female table or Fid to Male table as their foreign key.

Male

Mid	Name	Age	Address

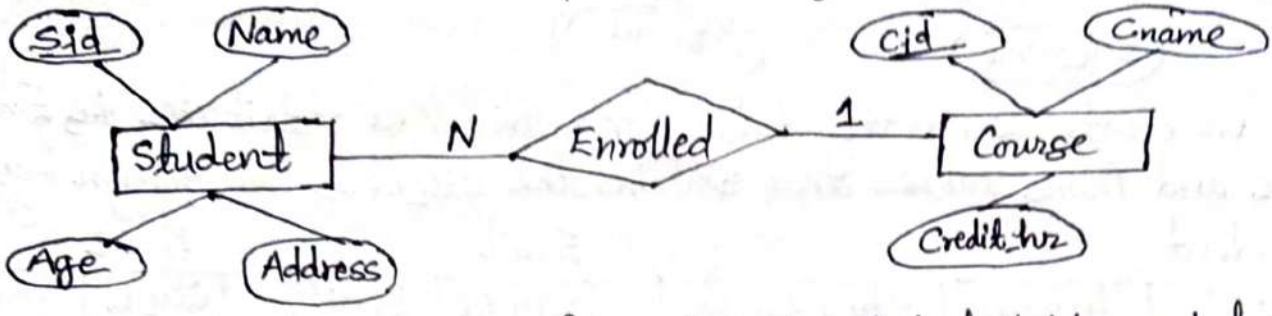
Female

Fid	Mid	Fname	No of Child

ii) Mapping of Binary 1:N relationship types to ER:

For binary one-to-many relationship identify the relation that represent the participating entity type at the N-side of the relationship type and then include primary key of one side entity set into many side entity set as foreign key. Seperate relation is created for the relationship set only when the relationship set has it's own attributes.

Example: Let's take ER diagram with many to one relationship as below;



Here we can set Cid as foreign key to Student table as below;

Student

Sid	Name	Age	Address	Cid

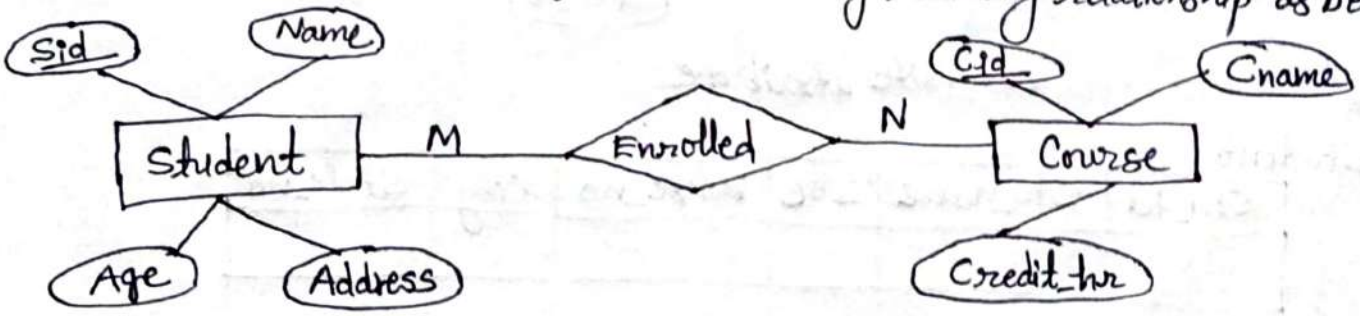
Course

Cid	Cname	Credit_hr

iii) Mapping of Binary M:N relationship types to ER:

For a binary many-to-many relationship type, seperate relation is created for the relationship type. Primary key for each participating entity set is included as foreign key in the relation and their combination will form the primary key of the relation.

Example:- Let's take ER diagram with many-to-many relationship as below;



Here, we can create a new table Enrolled that contains the primary keys of entities student and course entities as below;

Student

Sid	Name	Age	Address

Course

Cid	Cname	Credit_hr

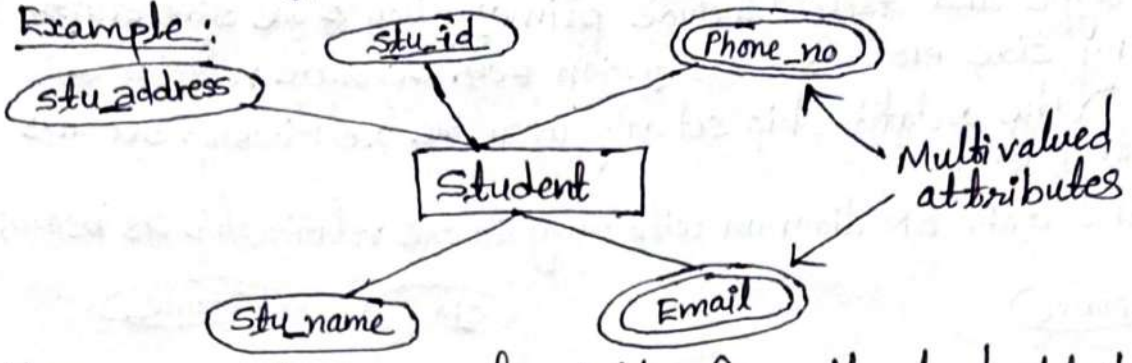
Enrolled

Sid	Cid

4. Mapping of multivalue attributes to ER:

If an entity has multivalued attribute, separate relation is created with primary key of the entity set and multivalued attribute itself.

Example:



Here we create separate table for multivalued attributes i.e, Email table and Phone table that contains the stu_id as their foreign key.

Student

Stu_id	Stu_name	Stu_address

Email

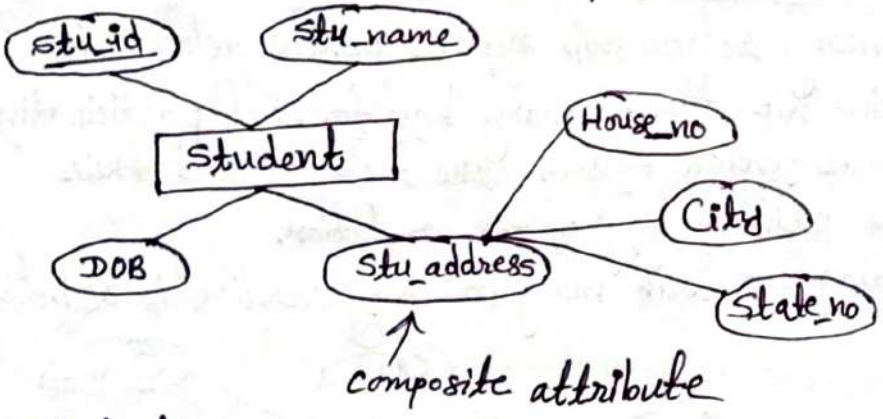
Stu_id	Email

Phone

Stu_id	Phone_no

5. Mapping composite attributes to ER:

If an entity has composite attributes, no seperate attribute (column) is created for composite attribute.



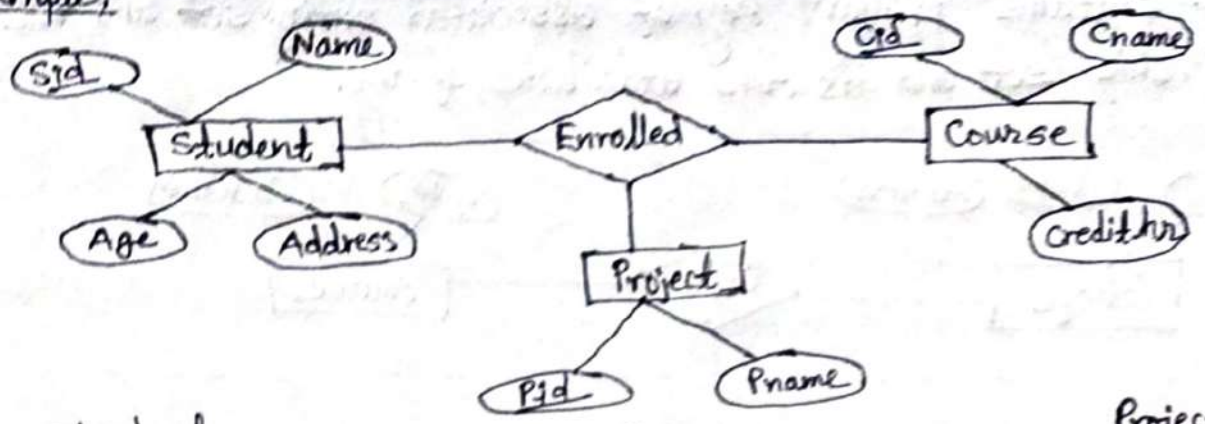
Student

Stu_id	Stu_name	DOB	House_no	City	State_no

6. Mapping of N-ary relationship types to ER:-

For each n-ary relationship set for $n>2$, a new relation is created. Primary keys of all participating entity sets are included in the relation as foreign key attributes. Besides this all simple attributes are included as attributes of the relation.

Example:



Student

Sid	Name	Age	Address

Course

Cid	Cname	credithr

Project

Pid	Pname

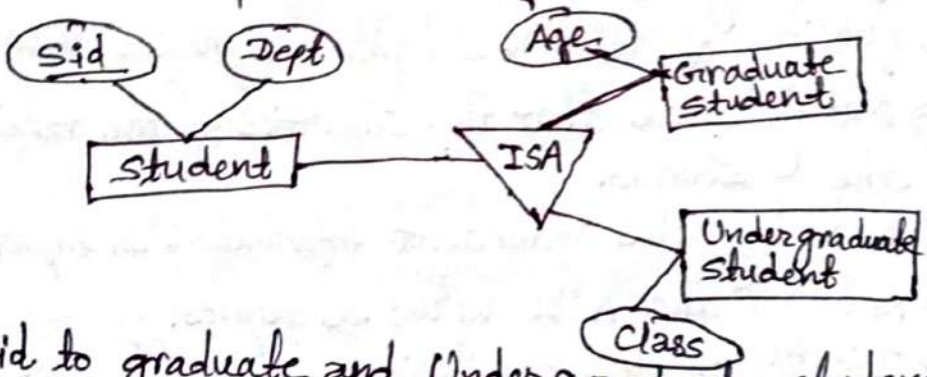
Enrolled

Sid	Cid	Pid

7. Mapping Specialization/generalization to ER:-

To construct relational table for this, we set primary key of the supper class to their sub classes as their foreign key. If subclasses are disjoint and complete then relation for a subclass entity set includes all attributes of superclass entity set and all of its own attributes.

Example:



Here we set sid to graduate and Undergraduate student tables as below;

Student

Sid	Dept

Graduate student

Sid	Age

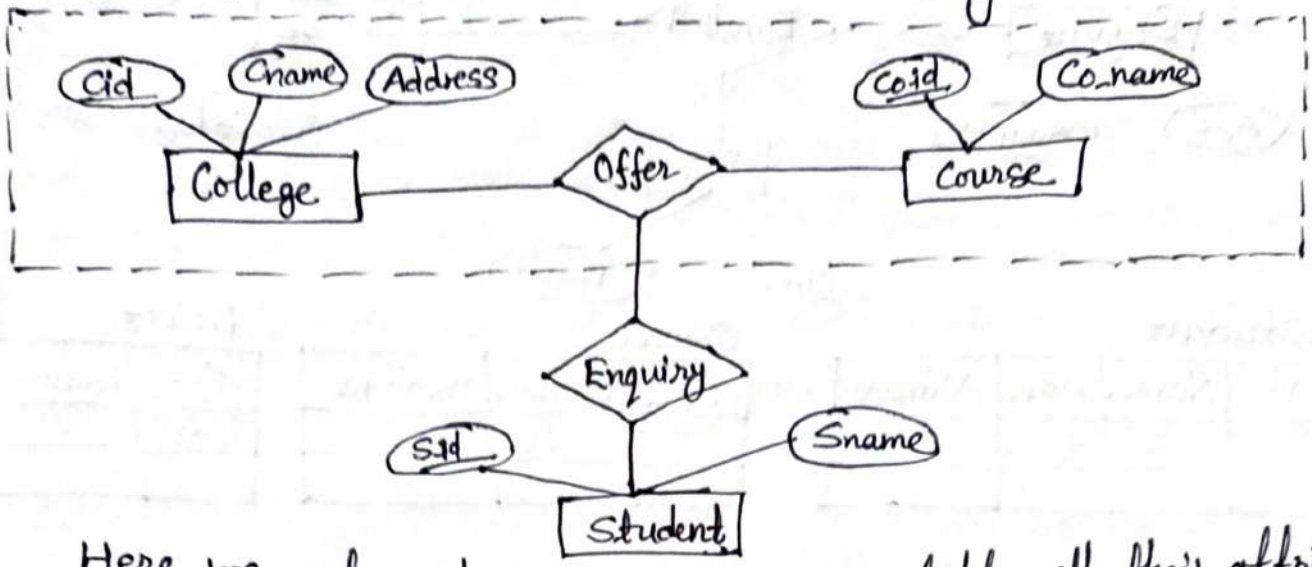
Undergraduate students.

Sid	Class

8. Mapping Aggregation to ER:-

In this there is no distinction between entity sets and relationship sets. in relational model. In relational model seperate relation is created for this relationship and the

relation contains primary key of associated entity set and the relationship set and its own attributes, if any.



Here we set cid and coid to enquiry table with their attributes;

College

Cid	Cname	Address

Course

Coid	Co_name

Enquiry

Sid	Sname	Cid	Coid

#Informal Design Guidlines for Relational Schemas:

Informal guidlines that may be used as measures to determine the quality of relation schema design as listed below:

→ Making sure that the semantics of the attributes is clear in the schema.

→ Reducing the redundant information in tuples.

→ Reducing the NULL values in tuples.

→ Disallowing the possibility of generating spurious tuples.

⊛. Imparting clear semantics to attributes in relations;

The semantics of a relation refers to it's meaning resulting from the interpretation of attribute values in a tuple. The relational schema design should have a clear meaning. We can thus formulate the following informal design guidline.

Guidline 1: Design a relation schema so that it is easy to explain it's meaning. Do not combine attributes from multiple

entity types into a single relation.

If a relation schema corresponds to one entity type or one relationship type, it is straight forward to explain it's meaning. Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

⊛ Redundant information in tuples and update anomalies:

One goal of schema design is to minimize the storage space used by the base relations (and hence the corresponding files). Grouping attributes into relation schemas has a significant effect on storage space.

Storing natural joins of base relations leads to an additional problem referred to as update anomalies. These are: insertion anomalies, deletion anomalies, and modification anomalies.

Insertion Anomalies happen;

→ When insertion of a new tuple is not done properly and will therefore can make the database become inconsistent.

→ When the insertion of a new tuple introduces a NULL value.

Deletion Anomalies happen;

→ When the insertion of a new tuple introduces a NULL value.

Modification Anomalies happen;

→ When we fail to update all tuples as a result in the change in a single one.

Guidline 2: Design the base relation schemas so that no insertion, deletion or modification anomalies are present in the relations.

If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

⊛. NULL values in Tuples;

If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples. This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level.

Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied. SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable.

Guidline 3: As much as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only.

⊛. Generation of Spurious Tuples;

A relation in which too many attributes are grouped is called fat relation. Often, we may elect to split a "fat" relation into two relations, with the intention of joining them together if needed. However, applying a NATURAL JOIN may not yield the desired effect. On the contrary, it will generate many more tuples and we cannot recover the original table.

Guidline 4: Design relation schemas so that they can be joined with equility conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated.

Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Functional Dependencies

meaning limitation बाधा

A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes A1, A2, ..., An. If we think of the whole database being described by a single universal relation schema R = {A1, A2, ..., An}.

<u>Definition:-</u> A functional dependency, denoted by X→Y, between two sets of attributes X and Y that are subsets of R, such that any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y]. This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component. We say that Y is functionally dependent on X.

Functional dependency is represented as **FD** or **f.d.** The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.

<u>Example:</u>

Employee number	Employee Name	Salary	City
1	Bipin	50000	Butwal
2	Nisha	40000	Pokhara
3	Ram	38000	Kathmandu

In this example, if we know the value of Employee number, we can obtain Employee Name, City, Salery. By this, we can say that the City, Employee Name, and Salary are functionally dependent on Employee number.

<u>Conditions:</u>

→ If a constraint on R states that there can not be more than one tuple with a given X-value in any relation instance r(R) – that is, X is a <u>candiate key</u> of R. If X is a candidate key of R, then X→R.

→ If X→Y in R, this does not say whether or not Y→X in R.

#Normal Forms Based on Primary Keys:

⊛ Normalization:

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. It can be considered as a "filtering" or "purification" process to make the design have successively better quality.

Hence normalization of data is a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of:-

→ minimizing redundancy and

→ minimizing the insertion, deletion and update anomalies.

We assume that a set of functional dependencies is given for each relation, and that each relation has a designated primary key. Each relation is then evaluated for adequacy and decomposed further as needed to achieve higher normal forms, using the normalization theory.

⊛. Normal Forms

The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized. Normal forms, when considered in isolation from other factors, do not guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is in a given normal form. Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

→ nonadditive join or lossless join property

→ dependency preservation property.

⊛ Practical Use of Normal Forms;

Most practical design projects acquire existing designs of databases from previous designs, designs in legacy models, or from existing files. Although several higher normal forms have been defined, database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.

The database designers need not normalize to the highest possible normal form. Relations may be left in a lower normalization status, such as 2NF, for performance reason.

Denormalization → It is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

⊛ Definitions of Keys and Attributes Participating in Keys:

Superkey → A superkey of a relation schema $R=\{A1, A2, \ldots, An\}$ is a set of attributes $S \subseteq R$ with the property that no two tuples $t1$ and $t2$ in any legal relation state r of R will have $t1[S] = t2[S]$.

Key → A key K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey anymore. The difference between a key and a superkey is that a key has to be minimal; that is, if we have a key

$K=\{A1, A2, \ldots, Ak\}$ of R, then $K-\{Ai\}$ is not a key of R for any Ai, $1 \leq i \leq k$.

Example: {Ssn} is a key for EMPLOYEE, where {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys.

Candidate key → If a relation schema has more than one key each.

One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondry keys. In practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default superkey. If ~~tabet~~ table EMPLOYEE contains {Ssn} as the only candidate key for EMPLOYEE, then it is also the primary key.

Prime attribute → An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R.

Non-Prime attribute → An attribute is called nonprime if it is not a member of some candidate key of R.

⊛. First Normal Form (1NF):-

A relation is in first normal form if it does not contain any composite or multi-valued attribute. If a relation contain composite or multi-valued attribute, it violates first normal form. Simply we can say that, A relation is in first normal form if every attribute in that relation is single valued attribute.

Example: Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. It's decomposition into 1NF has been shown in table 2.

STUD_NO	STU_NAME	STUD_PHONE	STUD_ADDRESS
1	RAM	9816271721 9871712717	Kathmandu
2	SURESH	9848162117	Butwal

Table 1

Conversion from table 1 to 1NF in table 2.

STUD_NO	STU_NAME	STUD_PHONE	STUD_ADDRESS
1	RAM	9816271721	Kathmandu
1	RAM	9871712717	Kathmandu
2	SURESH	9848162117	Butwal

Table 2

Partial Depencency = If Proper subset of candidate key determines non-prime attribute

Second Normal Form (2NF):

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency. i.e, non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Partial Dependency → If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Example: Consider table as below;

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	10000
2	C2	15000
1	C4	20000
4	C3	10000
4	C1	10000
2	C5	20000

Note that ~~many~~ ~~way~~ there are many courses having the same course fee

Here,
COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;
COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;
COURSE-FEE together with COURSE_NO cannot decide the value of STUD_NO;

Hence,
COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate {STUD_NO, COURSE_NO};
But, COURSE_NO → COURSE_FEE i.e, COURSE_FEE is dependent on COURSE_NO, which is a proper subset of candidate key.
Non-prime attribute COURSE_FEE is dependent on a proper subset of the candiate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF:

we need to split the table into two tables such as;

Table 1: STUD_NO, COURSE_NO

Table 2: COURSE_NO, COURSE_FEE

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1
2	C5

Table 2

COURSE_NO	COURSE_FEE
C1	10000
C2	15000
C3	10000
C4	20000
C5	20000

Note: 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we do not need to store it's fee 10000 for all the 100 records, instead we can store it in the second table only once.

Third Normal Form (3NF):

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form. A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X→Y.

i) X is a super key.

ii) Y is a prime attribute (each element of Y is part of some candidate key).

Transitive dependency → If A→B and B→C are two FDs then A→C is called transitive dependency.

Example:

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

In above table;

FD set: {STUD_NO→STUD_NAME, STUD_NO→STUD_STATE, STUD_STATE→STUD_COUNTRY, STUD_NO→STUD_AGE}

Candidate key: {STUD_NO}

For this relation, STUD_NO→STUD_STATE and STUD_STATE→STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form.

To convert it in third normal form, we will decompose the relation STUDENT as:

STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE_COUNTRY (STATE, COUNTRY).

Boyce-Coded Normal Form (BCNF):

BCNF was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF.

Definition: A relation schema R is in BCNF if whenever a nontrivial function dependency $X \rightarrow A$ holds in R, then X is a superkey of R. In practice, most relation schemas that are in 3NF are also in BCNF, if for every FD, LHS is a super key.

BCNF is free from redundancy. If a relation is in BCNF, then 3NF is also satisfied. Every Binary Relation (a relation with only 2 attributes) is always in BCNF. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE Table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table functional dependencies are as follows:

EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

& Candidate key: {EMP_ID, EMP_DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys

To convert the given table into BCNF, we decompose it into three tables: EMP_COUNTRY, EMP_DEPT and EMP_DEPT_MAPPING as follows:

EMP_COUNTRY Table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT Table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional Dependencies:

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Now this is in BCNF because left side part of both the functional dependencies is a key.

#Properties of Relational Decomposition

1) Dependency Preservation:

If each functional dependency X→Y specified in FD appears directly in one of the relation schemas Ri in the decomposition or could be inferred from the dependencies that appear in some Ri. This is the Dependency Preservation. If a decomposition is not dependency preserving some dependency is lost in decomposition. To check this condition, take the JOIN of 2 or more relations in the decomposition.

For Example:-

R = (A, B, C)

FD = {A→B, B→C}

Key = {A}

R is not in BCNF.

Decomposition R1 = (A, B), R2 = (B, C).

R1 and R2 are in BCNF, lossless-join decomposition, Dependency preserving. Each functional dependency specified in FD either

appears directly in one of the relations in the decomposition. It is not necessary that all dependencies from the relation R appear in some relation Ri.

2) Lossless Join:

Lossless join is the ability to ensure that any instance of the original relation can be identified from corresponding instances in the smaller relations.

For Example:

Let R: relation, FD: set of functional dependencies on R,
X, Y: decomposition of R,

A decomposition {R1, R2, ..., Rn} of a relation R is called a lossless decomposition for R if the natural join of R1, R2, ..., Rn produces exactly the relation R.

A decomposition is lossless if we can recover:

R(A, B, C) → Decompose → R1(A, B) R2(A, C) → Recover → R'(A, B, C)

Thus R' = R.

Decomposition is lossless if:

X intersection Y → X (i.e, all attributes common to both X and Y functionally determine all the attributes in X.)

X intersection Y → Y (i.e, all attributes common to both X and Y functionall determine all the attributes in Y.)

If X intersection Y forms a superkey of either X or Y, the decomposition of R is a lossless decomposition.

Multivalued Dependency

Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on third attribute. A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors (white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M 3001	2013	Black
M 4006	2017	White
M 4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below;

BIKE_MODEL →→ MANUF_YEAR

BIKE_MODEL →→ COLOR.

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

⊛. Concept of Fourth normal form (4NF):

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Properties: A relation R is in 4NF if and only if the following conditions are satisfied:

i) It should be in the Boyce-Codd Normal Form (BCNF).

ii) The table should not have any Multi-valued dependency.

A table with a multivalued dependency violates the normalization standard of 4NF, because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this upto 4NF, it is necessary to break information into two tables.

Unit=8

Introduction to Transaction Processing Concepts and Theory:

⊛. Single-user versus multiuser system:

Characteristics	Single-user system	Multi-user system
Definition	i) A single-user system is a system in which only one user can access the computer system at a time.	i) A multi-user system is a system that allows more than one user to access a computer system at one time.
Super User	ii) A super user gets all the powers of maintaining the system and making changes to ensure the system runs smoothly.	ii) Super user does not exist when it comes to a multi-user system as each entity has control over their working.
Complexity	iii) Single-user system is simple.	iii) Multi-user system is complex.
Performance	iv) Only one task at a time gets performed.	iv) Shedules different tasks for performance at any rate.
Example	v) Windows, Apple Mac OS.	v) UNIX, LINUX

⊛. Transactions, Database Items, Read and Write Operations & DBMS Buffers

A transaction is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations – these can include insertion, deletion, modification, or retrieval operations. The database operations that form a transaction can either be embedded within an application program or they can be specified via a high-level query language such as SQL.

If the database operations in a transaction do not update the database but only retrive data, the transaction is called a read-only transaction; otherwise it is known as read-write transaction.

Database Item (Data item):

A database is basically represented as a collection of named data items. The size of a data item is called its granularity. A data item can be a database record, but it can also be a larger unit such as a whole disk block, or even a smaller unit such as an individual field (attribute) value of some record in the database.

Each data item has a unique name, but this name is not typically used by the programmer; rather, it is just a means to uniquely identify each data item.

Read and Write Operations:

The basic datatabse access operations that a transaction can include ~~as~~ are as follows;

i) read_item(X) → Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also ~~m~~ named X.

ii) write_item(X) → Writes the value of program variable X into the database item named X.

Executing the read_item(X) command includes following steps:

→ Find the address of disk block that contains item X.
→ Copy the disk block into a buffer in main memory (if that disk block is not already in some main memory buffer). The size of the buffer is the same as the disk block size.
→ Copy item X from the buffer to the program variable named X.

Executing a write_item(X) command includes following steps:

→ Find the address of the disk block that contains item X.
→ Copy the disk block into a buffer in main memory.
→ Copy item X from the program variable named X into its correct location in the buffer.
→ Store the updated disk block from the buffer back to disk (either immediately or at some later point in time).

DBMS Buffers:

A database buffer is a temporary storage area in the main memory. It allows storing the data temporarily when moving from one place to another. A database buffer stores a copy of disk blocks. But, the version of block copies on the disk may be older than the version in the buffer.

The DBMS will maintain in the database cache a number of data buffers in main memory. Each buffer typically holds the contents of one database disk block, which contains some of the database items being processed. When these buffers are all occupied, and additional database disk blocks must be copied into memory, some buffer replacement policy is used to choose which of the current occupied buffers is to be replaced. Some commonly used buffer replacement polices are LRU (least recently used).

⊛. Why do we need concurrency control?

⇒ Concurrency Control is the management procedure that is required for controlling existing execution of the operations that take place on a database. It is a procedure of managing simultaneous operations without conflicting with eachother. We need concurrency control method in DBMS for following reasons:

→ To apply isolation through mutual exclusion between conflicting transactions.

→ To resolve read-write and write-write conflict issues.

→ To preserve database consistency through constantly preserving execution obstructions.

→ The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.

→ Concurrency control helps to ensure serializability.

⊛. Why do we need recovery?

⇒ Database systems, like any other computer system, are subject to failures but the data stored in it must be available when required. When a database fails it must

passess the facilities for fast recovery. It must also have atomicity i.e, either transactions are completed successfully and committed or the transaction should have no effect on the database. So, to prevent data loss recovery techniques based on immediate update or backing up data can be used. The techniques used to recover the lost data due to system crash, transaction errors, viruses, incorrect command execution etc. are database recovery techniques.

⊛ Transaction States and Operations:

A transaction is an atomic unit of work that should either be completed in it's entirety or not done at all. For recovery purposes, the system needs to keep track of when each transaction starts, terminates and aborts. Therefore the recovery manager of the DBMS needs to keep track of the following operations: (i.e, transaction operations)

BEGIN_TRANSACTION → This marks the beginning of transaction execution.

READ or WRITE → These specify read or write operations on the database items that are executed as part of a transaction.

END_TRANSACTION → This specifies the READ and WRITE transaction operations have ended and marks the end of transaction execution.

COMMIT_TRANSACTION → This signals a successful end of the transaction so that any updates executed by the transaction can be safely committed to the database and will not be undone.

ROLLBACK (or ABORT) → This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Recovery techniques use the following operators:

UNDO → Similar to rollback except that it applies to a single operation rather than to a whole transaction.

REDO → This specifies that certain transaction operations must be redone to ensure that all the operations of a commited transaction have been applied successfully to the database.

Transaction States:-

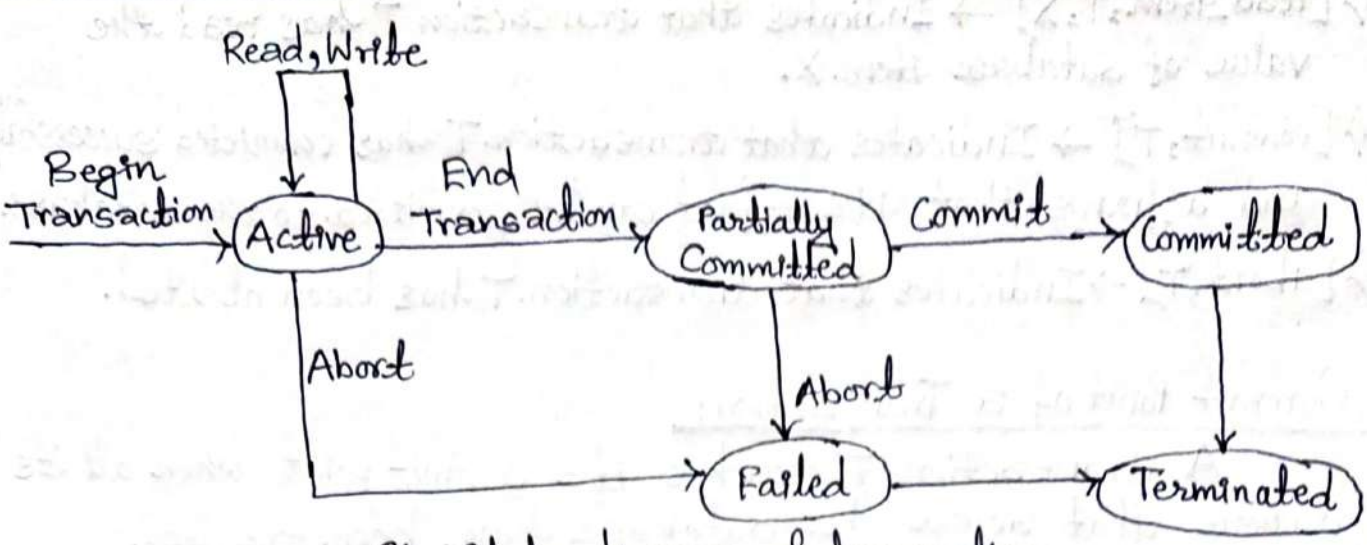


fig. State diagram of transaction.

The above diagram illustrates that how a transaction moves through its ~~executio~~ execution states. A transaction goes into an active state immediately after it starts execution, where it can execute READ and WRITE operations. When transaction ends, it moves to the partially commited state. Once some recovery protocols are ensured, the transaction is said to have reached its commit point and enters the committed state. Finally it has concluded its execution successfully and all it's changes must be recorded permanently in the database, even if a system failure occurs.

⊛. The System log:

The log is a sequential, append-only file that is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure. One or more main memory buffers, called the log buffers, hold the last part of the log file. The following are the types of entries – called log records – that are written to the log file and the corresponding action for each log record. In these entries, T refers to a unique transaction_id that is generated automatically by the system for each transaction and that is used to identify each transaction:

i) [start_transaction, T]. → Indicates that transaction T has started execution.

ii) [write_item, T, X, odd_value, new_value]. → Indicates that transaction T has changed the value of database item X from odd_value to new_value.

iii) [read_item, T, X] → Indicates that transaction T has read the value of database item X.

iv) [commit, T] → Indicates that transaction T has completed successfully and affirms that it's effect can be committed to the database.

v) [abort, T] → Indicates that transaction T has been aborted.

⊛. Commit Point of a Transaction:

A transaction T reaches it's commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database has been recorded in the log. Beyond the commit point, the transaction is said to be committed, and it's effect is assumed to be permanently recorded in the database. The transaction then writes an entry [commit, T] into the log.

If a system failure occurs we can search back in the log for all transactions T that have written a [start_transaction, T] record into the log but have not written their [commit, T] record yet. These transactions may have to be rolled back to undo their effect on the database during the recovery process. Transactions that have written their commit record in the log must also have recorded all their WRITE operations in the log, so their effect on database can be redone from the log records. Notice that the log file must be kept on disk because at the time of crash, the contents of main memory may be lost.

⊛. Buffer replacement polices:

If all the buffers in the DBMS cache are occupied and new disk pages are required to be loaded into main memory from disk, a page replacement policy is needed to select the particular buffers to be replaced. Some page replacement polices that have been developed specifically for database systems are as follows:-

1) Domain Seperation (DS) Method → In this method, the DBMS cache is divided into seperate domains (set of buffers). Each domain handles one type of disk pages, and page replacements within each

domain are handled via the basic LRU (least recently used) page replacement. This achieves better performance on average but it does not adopt to dynamically changing loads because the number of available buffers for each domain is predetermined.

ii) Hot Set Method → This algorithm is useful in queries that have to scan a set of pages repeatedly, such as when a join operation is performed using the ~~tes~~ nested-loop method. This algorithm does not replace disk pages until their processing is completed.

iii) The DBMIN Method → This policy uses a model known as QLSM (query locality set model), which predetermines the pattern of page refrences for each algorithm for a particular type of database operation.

⊛. Desirable properties of transactions:-

Transactions should possess several properties, often called the ACID properties. The following are the ACID properties;

i) Atomicity → A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.

ii) Consistency preservation → A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

iii) Isolation → A transaction should not make its updates visible to other transactions until it is committed. That is, the execution of a transaction should not be interfered by any other transactions executing concurrently.

iv) Durability or permanency → The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

⊛ Schedules (Histories) of Transactions:-

A schedule (or history) S of n transactions $T_1, T_2, \ldots, T_n$ is an ordering of the transactions. The transactions executing

concurrently in an interleaved fashion, from various transactions forms is known as transaction schedule. For each transaction T_i that participates in the schedule S, the operations of T_i in S must appear in the same order in which they occur in T_i. The order of operations in S is considered to be a total ordering, meaning that for any two operations in the schedule, one must occur before the other.

⊗. Characterizing schedules based on recoverability:

For some schedules it is easy to recover from transaction and system failures, whereas for other schedules the recovery process can be quite involved. In some cases, it is even not possible to recover correctly after a failure. Hence, it is important to characterize the types of schedules for which recovery is possible, as well as for which recovery is relatively simple. These characterizations do not actually provide the recovery algorithm; they only attempt theoretically characterize the different types of schedules.

i) Recoverable schedule → One where no transaction needs to be rolled back. A schedule S is recoverable if no transaction T in S commits until all transactions T that have written an item that T reads have committed.

ii) Cascadeless schedule → One where every transaction reads only the items that are written by committed transactions.

iii) Schedules requiring cascaded rollback → A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back.

iv) Strict Schedules → A schedule in which a transaction can neither read nor write an item X until the last transaction that wrote X has committed.

#Characterizing Schedules Based on Serializability:-

The concept of serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules. This section defines serializability and discuss how it may be used in practice. A serializable schedule is of two types: Conflict Serializable and view serializable.

i) Conflict Serializable → A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. Two operations are said to be conflicting if all conditions satisfy:

→ They belong to different transactions
→ They operate on the same data item
→ At least one of them is a write program.

ii) View Serializable → A schedule is called view serializable if it's view equals to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability contains blind writes, then the view serializable does not conflict serializable.

⊛. Testing for conflict serializability of a schedule:

Algorithm for testing conflict serializability of a schedule S:

1. Looks at only read_Item(X) and write_Item(X) operations.
2. Constructs a precedence graph – a graph with directed edges.
3. An edge is created from T_i to T_j if one of the operation in T_i appears before a conflicting operation in T_j.
4. The schedule is serializable if and only if the precedence graph has no cycles.

⊛. How serializability is used for concurrency control:

Ans:- A concurrency control is said to be serializable if the final result of that schedule is totally same as the final result given by the serial schedule. So in executing transaction it will be allowed to access multiple transaction with the DBMS in orders to enhance the efficiency and the maximum usability of the resources. But it should give the same result for a particular transaction, otherwise it is useless. So through serializability it will lead to the same final outcome.

UNIT=9

Concurrency Control Techniques

Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with eachother. Different concurrency control protocols/techniques offer different benefits between the amount of concurrency they allow and the overhead they impose.

1) Two-Phase Locking Technique:

for concept, if felt lengthy can be escaped.

A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. Generally, there is one lock for each data item in the database.

Locking is an operation which secures: permission to read, or permission to write a data item. Two phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock. The 3 activities taking place in the two phase update algorithm are:-

→ lock acquisition
→ Modification of data
→ Release lock

Two phase locking prevents deadlock from occuring in distributed systems by releasing all the resources it has acquired, if it is not possible to acquire all the resources required without waiting for another process to finish using a lock. A transaction in the Two Phase Locking Protocol can assume one of the two phases:

i) Growing Phase → In this phase a transaction can only acquire locks but cannot release any lock. The point when a transaction acquires all the locks it needs is called the lock Point.

ii) Shrinking Phase → In this phase a transaction can only release locks but cannot acquire any.

The two main modes in which a data item may be locked are;

i) exclusive (X) mode → Data item can be both read as well as written.

ii) Shared (S) mode → Data item can only be read.

	S	X
S	True	False
X	False	False

Fig. Lock-compatibility Matrix

⊛ Types of locks and System lock tables:

i) Binary Locks → A binary lock can have two states or values: locked and unlocked (or 1 and 0, for simplicity). A distinct lock is associated with each database item X. If the value of the lock on X is 1, item X cannot be accessed by a database operation that requests the item. If the value of the lock on X is 0, the item can be accessed when requested, and the lock value is changed to 1. We refer to the current state of the lock associated with item X as lock(X). Two operations, lock_item(X) and unlock_item(X) are used with binary locking.

ii) Shared/Exclusive (or Read/Write) Locks → In this lock there are three locking operations: read_lock(X), write_lock(X) and unlock(X). A lock associated with an item X, LOCK(X), now has three possible states: read-locked, write-locked, or unlocked. A read-locked item is also called share-locked because other transactions are allowed to read the item, whereas a write-locked item is called exclusive-locked because a single transaction exclusively holds the lock on the item.

⊛. Conversion (Upgrading, Downgrading) of Locks

A transaction that already holds a lock on item X is allowed under certain conditions to convert the lock from one locked state to another. For example, it is possible for a transaction T to issue a read_lock(X) and then later to upgrade the lock by issuing a write_lock(X) operation. Similarly it is also possible for a transaction T to issue a write_lock(X) and then later to downgrade the lock by issuing a read_lock(X) operation. When upgrading and downgrading of locks is used, the lock table must include transaction identifiers in the record structure for each lock to store the information on which transactions hold locks on the item.

⊛. Guaranteeing Serializability by Two-Phase Locking:

A transaction is said to follow the two-phase locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction. Such a transaction can be divided into two phases: an expanding or growing phase, during which new locks on items can be acquired but none can be released; and a shrinking phase during which existing locks can be released but no new locks can be acquired. If lock conversion is allowed, then upgrading of locks must be done during the expanding phase and downgrading of locks must be done in the shrinking phase.

⊛. Basic, Conservative, Strict and Rigorous Two-Phase Locking:

OR Categories of two-phase locking (2-PL):

i) Basic 2-PL → Two-phase locking that we studied before is the basic 2-PL.

ii) Strict 2-PL → This requires in addition to basic 2-PL that all Exclusive (X) locks held by the transaction be released until after the transaction commits. Following strict 2-PL ensures that our schedule is: Recoverable and Cascadeless. Hence, it gives us freedom from Cascading Abort which was in Basic 2-PL, but still deadlocks are possible.

iii) Rigorous 2-PL → This requires in addition to basic 2-PL that all Exclusive (X) and Shared (S) locks held by the transaction be released until after the transaction commits. The difference between Strict 2-PL and Rigorous 2-PL is that, Rigorous is more restrictive, it requires both Exclusive and Shared locks to be held until after transaction commits.

iv) Conservative 2-PL → This requires to lock all the items it access before the transaction begins execution by predeclaring its read-set and write-set. If any of the predeclared items needed cannot be locked, the transaction does not lock any of the items, instead it waits until all the items are available for locking.

⊗. Dealing with Deadlock and Starvation:

Deadlock and Starvation both are the conditions where the processes requesting for a resource has been delayed for a long. Although deadlock and starvation both are different from each other in many aspects. Deadlock is a condition where no process proceeds for execution and each waits for resources that have been acquired by the other processes. On the other hand, Starvation is a condition where process with higher priorities continuously uses the resources preventing low priority process to acquire the resources.

Dealing with Deadlock:

One way to prevent deadlock is to use a deadlock prevention protocol. A number of other deadlock prevention schemes have been proposed. Some of these techniques use the concept of transaction timestamp, which is a unique identifier assigned to each transaction. An alternative approach to deal with deadlock is deadlock detection, where the system checks if a state of deadlock actually exists. This solution is attractive if different transactions will rarely access the same items at the same time. Timeouts is the another scheme to deal with deadlock. This method is practical because of its low overhead and simplicity. In this method, if a transaction waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it – regardless of whether a deadlock actually exists.

Dealing with Starvation: One solution for starvation is to have a fair waiting scheme, such as using a first-come-first-served queue; where transactions are enabled to lock an item in order in which they originally requested to lock. Another scheme allows some transactions to have higher priority over others. The wait-die and wound-wait avoid starvation, because they restart a transaction that have been aborted.

Deadlock vs Starvation

Deadlock	Starvation
i) Deadlock is a condition where no process proceeds for execution and each ~~wits~~ waits for resources that have been acquired by other processes.	i) Starvation is a condition where process with higher priorities continiously uses the resources preventing low priority process to acquire the resources.
ii) It happens if two or more transaction is waiting for each other.	ii) It happens if the waiting scheme for locked items is in unfair.
iii) Avoidance: ~~→Switch priorities so that every~~ → Acquire locks at once before starting. → Acquire locks with predefined order.	iii) Avoidance: → Switch priorities so that every thread has a chance to have high priority. → Use FIFO order among competing request.
iv) Deadlock is also known as circular waiting.	iv) Starvation is also known as lived lock.
v) Resources are blocked by the processes.	v) Resources are continiously utilized by high priority processes.

2) Timestamp Ordering:

A timestamp is a unique identifier created by the DBMS to identify a transaction. Timestamp Ordering is a schedule in which the transactions participate is then serializable, and the only equivalent serial schedule permitted has the transactions in order of their timestamp values. A timestamp can be implemented in 2 ways. One is to directly assign the current value of the clock to the transaction or data item. The other is to attach the value of a logical counter that keeps increment as new timestamps are required. The timestamp of a data item can be of 2 types:

i) W-timestamp(X) → This means the latest time when the data item X has been written into.

ii) R-timestamp(X) → This means the latest time when the data item X has been read from.

⊛. Basic Timestamp Ordering:

Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, then this algorithm compares the timestamp of T with R_TS(X) and W_TS(X) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp.

1. Check the following condition whenever a transaction T_i issues a read_item(X) operation:
   - → If W_TS(X) > TS(T_i) then the operation is rejected.
   - → If W_TS(X) <= TS(T_i) then the operation is executed.
   - → Timestamps of all the data items are updated.
2. Check the following condition whenever a transaction T_i issues a write_item(X) operation:
   - → If TS(T_i) < R_TS(X) then the operation is rejected.
   - → If TS(T_i) < W_TS(X) then the operation is rejected and T_i is rolled back otherwise the operation is executed.

where,
TS(T_i) denotes the timestamp of transaction T_i.
R_TS(X) denotes the Read timestamp of data-item X.
W_TS(X) denotes the Write timestamp of data-item X.

⊛. Strict Timestamp Ordering:

A variation of basic TO called strict TO (timestamp ordering) ensure that the schedules are both strict (for easy recoverability) and (conflict) serializable.

1. Transaction T issues a write_item(X) operation:
   - → If TS(T) > R_TS(X), then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).
2. Transaction T issues a read_item(X) operation:
   - → If TS(T) > W_TS(X), then delay T until the transaction T' that wrote or read X has terminated (committed or aborted).

⊛. Thoma's Write Rule:

A modification of a basic TO algorithm, known as Thoma's write rule, does not enfore conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(X) operation as follows;

1. If R_TS(X) > TS(T) then abort and roll-back T and reject the operation.
2. If W_TS(X) > TS(T), then just ignore the write operation and continue execution. This is because the most recent writes counts in case of two consecutive writes.
3. If the conditions given in 1 and 2 above do not occur, then execute write_item(X) of T and set W_TS(X) to TS(T).

3. Multiversion Concurrency Control:

These concurrency control keep copies of the old values of a data item when the item is updated. They are known as multiversion concurrency control because several versions (values) of an item are kept by the system. When a transaction requests to read an item, the appropriate version is chosen to maintain the serializability of the currently executing schedule. When a transaction writes an item, it writes a new version and the old version(s) of the item is retained. An obvious drawback of multiversion techniques is that more storage is needed to maintain multiple versions of the database items.

⊛ Multiversion technique based on timestamp ordering:

In this method, several versions $X_1, X_2, \ldots, X_k$ of each data item X are maintained. For each version, the value of version X_i and the following two timestamps associated with version X_i are kept:

i) read_TS(X_i) → The read timestamp of X_i is the largest of all the timestamps of transactions that have successfully read version X_i.

ii) write_TS(X_i) → The write timestamp of X_i is the timestamp of the transaction that wrote the value of version X_i.

To ensure serializability, the following rules are used:

i) If transaction T issues a write_item(X) operation, and version i of X has the highest write_TS(X_i) of all versions of X that is also less than or equal to TS(T), and read_TS(X_i) > TS(T), then abort and roll back transaction T; otherwise, create a new version X_j of X with read_TS(X_j) = write_TS(X_j) = TS(T).

ii) If transaction T issues a read_item(X) operation, find the version i of X that has the highest write_TS(X_i) of all versions of X that is also less than or equal to TS(T); then return the value of X_i to transaction T, and set the value of read_TS(X_i) to the larger of TS(T) and the current read_TS(X_i).

⊛. Multiversion locking using certify locks:- (OR Multi-version 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 read-locked, write-locked, certify-locked or unlocked. The idea behind multiversion 2PL is to allow other transactions T to read an item X while a single transaction T holds a write lock on X. This is accomplished by allowing two versions for each item X; one, the committed version, must always have been written by some committed transaction. The second local version X' can be created when a transaction T acquires a write lock on X.

Once T is ready to commit, it must obtain a certify lock on all items that it currently holds write locks on before it can commit. The certify lock is not compatible with read locks, so the transaction may have to delay it's commit until all its write-locked items are released by any reading transactions in order to obtain the certify locks. Once the certify locks are 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.

4. Validation (Optimistic) Techniques:

The optimistic approach is based on the assumption that the majority of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through 2 or 3 phases, referred to as read, validation and write.

i) Read phase → During read phase, the transaction reads the database, executes the needed computations and makes the updates to a private copy of the database values. All update operations of the transactions are recorded in a temporary update file, which is not accessed by the remaining transactions.

ii) Validation phase → During validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to write phase. If the validation test is negative, the transaction is restarted and the changes are discarded.

iii) Write phase → During write phase, the changes are permanently applied to the database.

⊛. Snapshot Isolation Concurrency Control:

In database and transaction processing, snapshot isolation is a guarantee that all reads made in a transaction will see a consistent snapshot of the database and the transaction itself will successfully commit only if no updates it has made conflict with any updates made since that snapshot.

Snapshot isolation has been adopted by several major database management systems, such as SQL, Oracle, MongoDB, Postgree SQL etc. The main reason for it's adoption is that it allows better performance than serializability. In practice snapshot isolation is implemented within multiversion concurrency control (MVCC), where generational values of each data item (versions) are maintained.

UNIT-10

Database Recovery Techniques

⊛. Recovery Outline and Categorization of Recovery Algorithms:

Recovery Outline: Recovery from transaction failures usually means that the database is restored to the most recent consistent state 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. A typical strategy for recovery may be ~~suzzn~~ summerized as follows:

i) If there is extensive damage to a wide portion of a database due to catastrophic failure, such as disk crash, the recovery method restores a past copy of database that was backed up.

ii) When the database on disk is not physically damaged, and a noncatastrophic failure has occurred, the recovery strategy is to identify any changes that may cause an inconsistency in the database.

Categorization of Recovery Algorithms:

1. Caching (Buffering) of disk blocks → In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. A collection of in-memory buffers called the DBMS cache is kept under control of DBMS for holding these buffers. A directory is used to keep track of which database items are in the buffer. A dirty bit is associated with each buffer, which is 0 if the buffer is not modified else 1 if modified.

2. Write-Ahead Logging → If the failure occurs, the RAM buffer that stores database information as well as the log may lost which will cause loosing of information. Write-Ahead logging protocol is derived to protect the system in such case. Write-Ahead logging states that;

i) The old value cannot be replaced by its new value until undo type logging record has been permanently stored in the disk.

ii) Prior to commit operation of a transaction, the redo portion and undo portion of the log have been written permanently in the disk. To make the recovery process more efficient DBMS recovery subsystem may maintain a list of transaction details, which include the list of active transactions that are not committed yet as well as the list of all the committed and aborted transactions until the last checkpoints.

3. <u>Steal/no-steal and force/no-force</u>: → These specify the rules that govern when a page from database cache can be written to disk:

→ If a cache buffer page updated by transaction cannot be written to disk before the transaction commits, the recovery method is called a no-steal approach. On the other hand, if the recovery protocol allows writing an updated buffer before the transaction commits, it is called steal.

→ If all pages updated by a transaction are immediately written to disk before the transaction commits, the recovery approach is called a force approach. Otherwise, it is called no-force.

4. <u>Checkpoints and Fuzzy Checkpointing</u> → Another type of entry in the log is called a checkpoint. It is a mechanism where all the previous logs are removed from the system and permanently stored in the storage disk. The checkpoint is like a bookmark. The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed. Taking a checkpoint consists of following actions:

i) Suspend execution of transactions temporarily.

ii) Force-write all main memory buffers that have been modified to disk.

iii) Write a [checkpoint] record to the log, and force-write the log to disk.

iv) Resume executing transactions.

To overcome delay transaction processing during the time needed to force-write all memory buffers, a technique is used called fuzzy checkpointing.

5. Transaction Rollback and Cascading Rollback:

If a transaction fails for whatever reason after updating the database, but before the transaction commits, it may be necessary to roll back the transaction. If any data item values have been changed by the transaction and written to the database on disk, they must be restored to their previous values (BFIMs). The undo-type log entries are used to restore the old values of data items that must be rolled back.

If a transaction T is rolled back, any transaction S that has, in the temprory, read the value of some data item X written by T must also be rolled back. Similarly, once S is rolled back, any transaction R that has read the value of some data item Y written by S must also be rolled back; and so on. This phenomenon is called cascading rollback, and it can occur when the recovery protocol ensures recoverable schedules but does not ensure strict or cascadeless schedules.

⊛ NO-UNDO/REDO Recovery Based on Deferred Update:

The idea behind deferred update is to defer or postpone any actual updates to the database on disk until the transaction completes its execution successfully and reaches its commit point. After the transaction reaches its commit point and the log is force-written to disk, the updates are recorded in the database.

If a 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. Therefore, only **REDO-type log entries** are needed in the log, which include the new value (AFIM) of the item written by a write operation. The **UNDO-type log entries** are not needed since no undoing of operations will be required during recovery.

A drawback of this method is, it limits the concurrent execution of transactions because all write-locked items remain locked until the transaction reaches its commit point. Additionaly, it may require excessive buffer space to hold all updated items until the transactions commit. This method's main benefit is that transaction operations never need to be undone, for two reasons:

i) A transaction does not record any changes in the database on disk until after it reaches its commit point.

ii) A transaction will never read the value of an item that is written by an uncommitted transaction.

⊛ Recovery Technique Based on Immediate Update:

In these techniques, when a transaction issues an update command, the database on disk can be updated immediately without any need to wait for the transaction to reach its commit point. It is not requirement that every update be applied immediately to disk; it is just possible that some updates are applied to disk before the transaction commits.

Provisions must be made for undoing the effect of update operations that have been applied to the database by a failed transaction. Theoretically, we can distinguish two main categories of immediate update algorithms:

i) If the recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is never a need to REDO any operations of committed transactions. This is called the UNDO/NO-REDO recovery algorithm. In this method, all

updates by a transaction must be recorded on disk before the transaction commits, so that REDO is never needed. Hence this method must utilize the steal/force strategy for deciding when updated main memory buffers are written back to disk.

ii) If the transaction is allowed to commit before all its changes are written to the database, we have the most general case, known as the UNDO/REDO recovery algorithm. In this case, steal/no-force strategy is applied.

⊛. Shadow Paging:

Shadow paging considers the database to be made up of a number of fixed sized disk blocks (say n) for recovery purposes. A directory with n entries is constructed, where the ith entry points to the ith database page on disk. The directory is kept in main memory if it is too large.

When a transaction begins executing, the current directory — whose entries point to the most recent or current database pages on disk — is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction.

During the transaction execution, the shadow directory is never modified. When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten. Instead, the new page is written elsewhere on some previously unused disk block. Figure below illustrates the concepts of shadow and current directories. For pages updated by the transaction two versions are kept. The old version is refrenced by the shadow directory and new version by the currend directory.

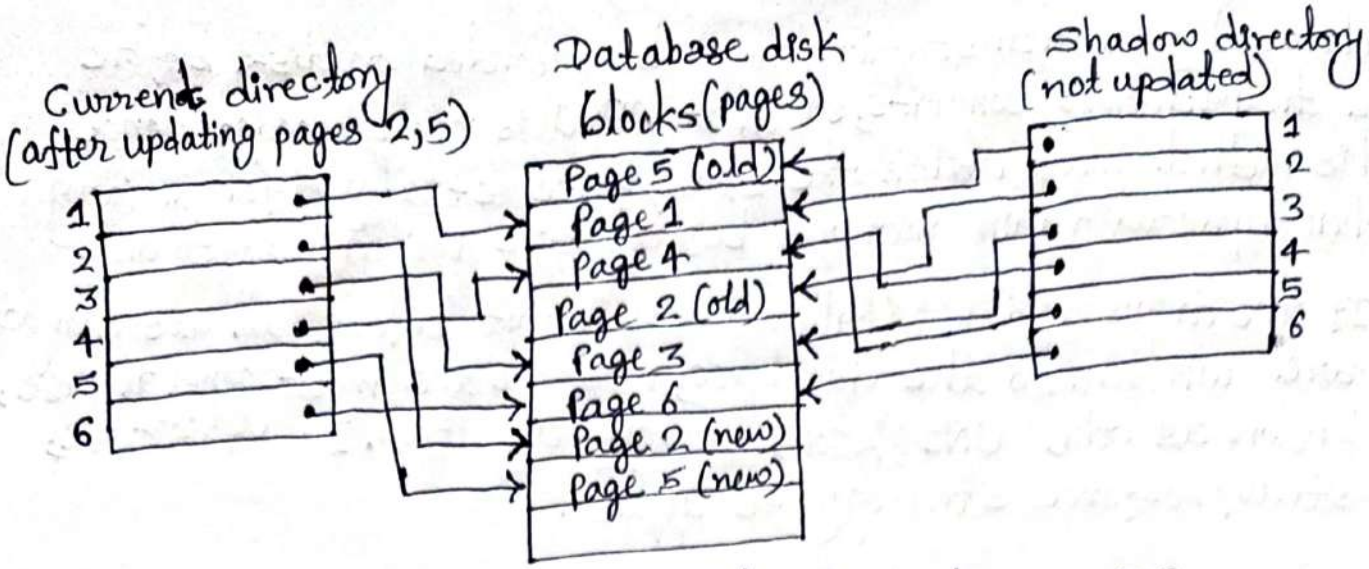


Fig. An example of shadow paging.

⊛ Database Backup and Recovery from Catastrophic Failures:

The recovery manager of a DBMS must be able to handle catastrophic failures such as disk crashes. The main technique to handle such crashes is a database backup, in which the whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes or other large capacity offline storage devices. In case of a catastrophic system failure, the latest backup copy can be reloaded from the tape to the disk, and the system can be restarted.

Data from critical applications such as banking, insurance, stock market and other databases is periodically backed up in its entirety and moved to physically seperate safe locations. To avoid losing all the effects of transactions that have been executed since the last backup, it is customary to back up the system log at more frequent intervals than full database backup by periodically copying it to magnetic tape.