

RESOURCE MATERIAL

ON

***DATABASE MANAGEMENT
SYSTEM***

(FOR 4TH SEMESTER CSE/IT)

Prepared by

Smt.Archana Tripathy

Lect.Comp.Sc.

Govt.polytechnic,Bhubaneswar.

CHAPTER-01

BASIC CONCEPTS OF DBMS

What is Database?

A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Computer databases typically contain aggregations of data records or files, containing information about sales transactions or interactions with specific customers.

What is DATABASE ?

Database

- A collection of related data with
 - * Logically coherent structure
 - * Inherent meaning
 - * Purpose, for intended users and applications
 - * Varying size
 - * Scope, content of varying breadth
 - * Physical organization of varying complexity
 - * Various applications with possibly-conflicting objectives
 - * Persistence, existence over a long period of time

Database Management System (DBMS):

It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications. Disadvantages in File Processing Data redundancy and inconsistency. Difficult in accessing data. Data isolation. Data integrity. Concurrent access is not possible. Security Problems.

DBMS:-

- ❖ A database system is nothing more than a computer based record keeping system i.e. whose purpose is to record and maintains the data or information.
- ❖ A DBMS is a software system that allows accessing data contained in a database.
- ❖ The objective of DBMS is to provide a convenient and effective method of defining, storing and retrieving information contained in the database.
- ❖ The DBMS interfaces with application programs so that the data contained in the database can be used by multiple applications and users.
- ❖ A database system involves four major components namely data, hardware, software and user.

- ❖ **Data:** - The fundamental unit of database is data. A database is therefore nothing but a repository for stored data. Every data must have two properties namely integrated and shared. Integrated means that the data can be uniquely identified in the database and shared means the data can be shared among several different users.
- ❖ **Hardware:** - The hardware consists of secondary storage volumes on which the database resides.
- ❖ **Software:** - Between the physical database and the users of the system there is a layer of software usually called as database management system (DBMS). All requests from users for access to the database are handled by DBMS.
- ❖ **Users:** - The users are the application programmers responsible for writing application programs that use the database. The application programmers operate on the data in all the usual ways, i.e. retrieving information, creating new information, deleting or changing existing information. The second classes of users are the end users whose task is to access the data of a database from a terminal. The end users use a query language to invoke user written application programs as per the commands from the terminal. The third classes of users are the database administrators or DBA who has control over the whole system.

1.1 PURPOSE OF DATABASE SYSTEMS:-

- Reduction of redundancies:-

- Centralized control of data by the Database administrator avoids unnecessary duplication of data and effectively reduces the total amount of data storage required. It also eliminates inconsistency and extra processing necessary to trace the data in large mass of data.

- Shared data:-

A database allows the sharing of data under its control by any number of application programs or users.

- Integrity:-

Centralized control can also ensure that adequate checks are incorporated in the DBMS to provide data integrity. Data integrity means that the data contained in the database is both accurate and consistent.

- Security:-

Data is of vital importance. These should not be accessed by unauthorized persons. DBMS can ensure that proper access procedures are followed including proper authentication schemes for access to DBMS and additional checks before permitting access to sensitive data. Different levels of security could be implemented for various types of data and operations.

- Conflict resolution:-

The conflicting requirements of various users and applications are solved by DBMS and best file structure and access methods are chosen to get optimal performance.

- Data independence:-

- DBMS supports both physical and logical data independence. Data independence is advantageous in the database environment since it allows for changes at one level of the database without affecting other levels.

Advantages of DBMS:

- Data Independence.
- Efficient Data Access.
- Data Integrity and security.
- Data administration.
- Concurrent access and Crash recovery.
- Reduced Application Development Time

APPLICATIONS OF DATABASE SYSTEMS:-

- Databases are widely used. Some of them are as follows.
- Banking-> For customer information, accounts, loans and banking transactions.
- Airlines-> For reservation and scheduled information.
- Universities-> For student, course and grade information.
- Credit card transactions-> For purposes on credit cards and generation of monthly statements.
- Telecommunications-> For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards and storing information about the communication network.
- Finance-> For storing information about holdings, sales and purchases of financial instruments such as stocks and bonds.
- Manufacturing-> For management of supply chain and for tracking production of items in factories, inventory of items in warehouses and orders for items.
- Human resources-> For information about employees, salaries, payroll taxes & benefits and for generation of payments

1.2 DATA ABSTRACTION (THREE LEVEL ARCHITECTURE OF DATABASE SYSTEM):-

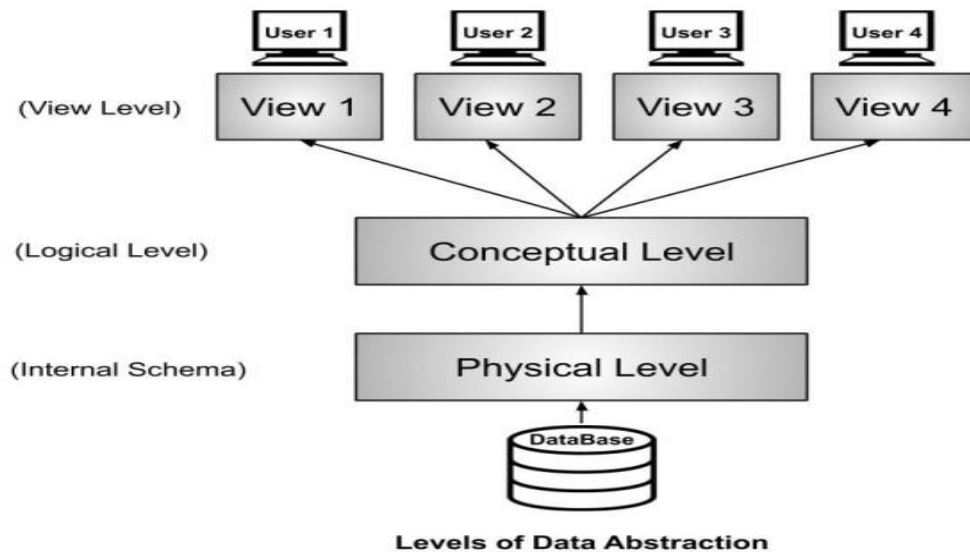
This architecture is used to provide a framework which is extremely useful in describing general database concepts and for explaining the structure of individual systems.

The purpose of designing a generalized database is so that it must have the capability to transform the query asked by the user into programming form so that the system can understand it and will be able to retrieve back the answer of the query.

It is divided into three levels.

- I. External level or view level or user view
- II. Conceptual level or logical level or global view
- III. Physical level or internal level or internal view

The view at each of these levels is described by a schema. A schema is an outline or a plan that describes the records and relationships existing in the view. The schema also describes the way in which entities at one level of abstraction can be mapped to next level.



1. Physical Level:-

The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low level data structure in detail. This level is expressed by physical schema which contains the definition of stored record, the method of representing the data fields and access aids used.

The internal level is the one closest to the physical storage. Whenever an external user query a database and the response of the query is available at the conceptual level then it is provided to the user. If the response is not available at the conceptual level then it is retrieved from the internal level.

2. Logical Level

1. The next higher level of abstraction describes what data are stored in the database and what relationship exists among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures.
2. It is defined by the logical schema. It describes all the records and relationships. There is one logical schema per database.
3. It also includes features that specify the checks to retain data consistency and integrity.

4. The conceptual schema is basically derived from the internal schema and it can be updated as per the demand of the users.

3. View Level:-

1. The highest level of data abstraction describes only part of the entire database. The system may provide many views for the same database.
2. Each view is described by a schema called external schema. The external schema consists of the definition of logical records and relationships in the view level.
3. The external schema also contains the method of deriving the objects in the external view from the objects in the conceptual view.
4. It is the level closest to the users i.e. it deals with the way in which data is viewed by the users.
5. All the types of database users use a particular language to query the database.
6. The three level architecture is designed in such a way that each and every level maintains their abstraction by their own. The DBMS controls all the levels and the DBMS is basically controlled by the DBA.

1.3 DATABASE USERS:-

There are five different types of database system users differentiated by the way they expect to interact with the system.

➤ **Naive Users:-**

- They are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

Example:- ATM user v The typical user interface for a naive user is a forms interface where the user can fill in appropriate fields of the form. Naive users may also simply read reports generated from the database.

➤ **Application Programmers:-**

- ❖ Application programmers are computer professionals who write application programs.
- ❖ Application programmers can choose from many tools like RAD tools, programming languages, fourth generation languages etc. to develop user interfaces.

➤ **Sophisticated Users:-**

- ❖ Sophisticated users interact with the system without writing programs. Instead they form their requests in database query language. They submit each such query to query processor whose function is to break down DML statements into instructions that the storage manager understands.
- ❖ Analysts who submit queries to explore data in the database fall in this category .

Database Administrator (DBA):-

- ❖ A person who has central control over the entire database system is called database administrator (DBA).
- ❖ The functions of DBA include
- ❖ Schema definition
- ❖ Storage structure and access method definition
- ❖ Schema and physical organization modification
- ❖ Granting of authorization for data access
- ❖ Routine maintenance

OR

Database Administrators (DBA):

The DBA is responsible for authorizing access to the database, for Coordinating and monitoring its use and for acquiring software and hardware resources as needed. These are the people, who maintain and design the database daily.

DBA is responsible for the following issues.

- a. Design of the conceptual and physical schemas: The DBA is responsible for interacting with the users of the system to understand what data is to be stored in the DBMS and how it is likely to be used. The DBA creates the original schema by writing a set of definitions and is permanently stored in the 'Data Dictionary'.
- b. Security and Authorization: The DBA is responsible for ensuring the unauthorized data access is not permitted. The granting of different types of authorization allows the DBA to regulate which parts of the database various users can access.
- c. Storage structure and Access method definition: The DBA creates appropriate storage structures and access methods by writing a set of definitions, which are translated by the DDL compiler.
- d. Data Availability and Recovery from Failures: The DBA must take steps to ensure that if the system fails, users can continue to access as much of the uncorrupted data as possible. The DBA also work to restore the data to consistent state

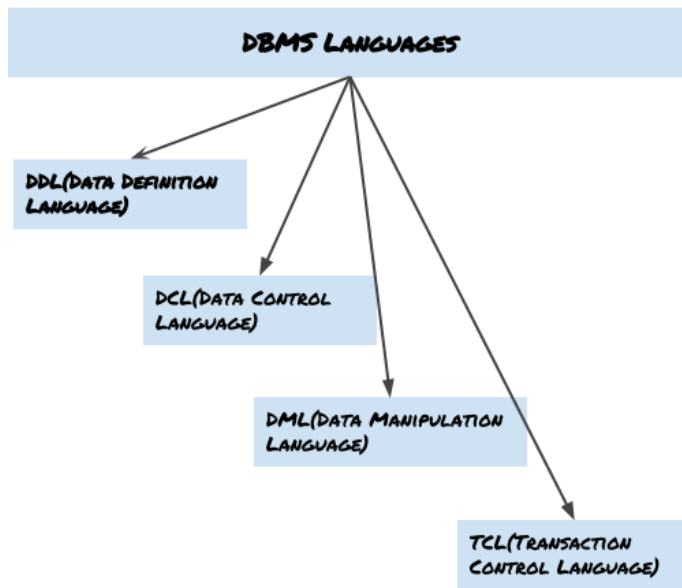
Database Designers:

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.

DATA BASE LANGUAGES:-

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

Types of Database Language



There are four types of database languages:-

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language (TCL)

1. Data Definition Language

- **DDL** stands for **Data Definition Language**. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for **Data Manipulation Language**. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

3. Data Control Language

- **DCL** stands for **Data Control Language**. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

(But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

1.5 DATA DICTIONARY:-

- Information regarding the structure and usage of data contained in the database, the metadata maintained in a data dictionary. The term system catalogue also describes this metadata. The data dictionary which is a database itself documents the data.

- Each database users can consult the data dictionary to learn what each piece of data and various synonyms of data fields mean.
- In an integrated system (i.e. in system where the data dictionary is a part of the DBMS) the data dictionary stores information concerning the external, conceptual and internal levels of the database. It contains the source of each data field value, the frequency of its use and an audit trail concerning updates, including the who and when of each update.

Model Questions

1.0 BASIC CONCPETS OF DBMS

1. Distinguish between Data and Information. (2)
2. Define Database system. (2)
3. What are the features of Database? (2)
4. What do you mean by Centralized data? (2)
5. Write the comparison between File management System and Database Management System .(6)
6. Define Data Abstraction Concept. (2)
7. What is data abstraction? What are the three level architecture of DBMS explain with diagram. (6)
8. What do you mean by Data redundancy? (2)
9. Name the different level of Data Abstraction. (2)
10. Define DBMS. What are the advantages and disadvantages of DBMS? (10)
11. Who is DBA? (2)
12. Data Dictionary (Short note) (4)
13. What are the roles of DBA? (6)
14. What are the roles & the responsibility of DBA? (6)
15. What do you mean by Data Independency? Explain its type. (6)
16. What do you mean by Data Independency? (2)
17. Name different types of database languages. (2)
18. Explain DML .Write the Syntax of CREATE, ALTER and DROP with Example. (10)

CHAPTER-02

DATA MODELS

2.1 Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

There are **two** types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.

2.2 E-R model

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modelling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.

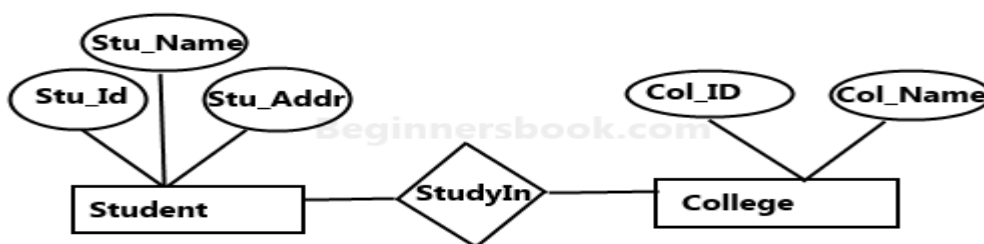
What is an Entity Relationship Diagram (ER Diagram)?

- An ER diagram shows the relationship among entity sets.

- An **entity set** is a group of similar entities and these entities can have attributes.
- In terms of DBMS, an **entity** is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Let's have a look at a simple ER diagram to understand this concept.

A simple ER Diagram:

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.



Sample E-R Diagram

Here are the geometric shapes and their meaning in an E-R Diagram. We will discuss these terms in detail in the next section(Components of a ER Diagram) of this guide so don't worry too much about these terms now, just go through them once.

Rectangle: Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set












Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

Double Ellipses: Multivalve Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set

	Represents Entity
	Represents Attribute
	Represents Relationship
	Links Attribute(s) to entity set(s) or Entity set(s) to Relationship set(s)
	Represents Multivalued Attributes
	Represents Derived Attributes
	Represents Total Participation of Entity
	Represents Weak Entity
	Represents Weak Relationships
	Represents Composite Attributes
	Represents Key Attributes / Single Valued Attributes

2.3 ENTITY SETS:-

- ❖ An entity is a thing or object in the real world that is distinguishable from all other objects. Example:- Each person of an enterprise.
- ❖ **An entity has a set of properties and values.** For some set of properties may uniquely identify an entity. Example:- addhar number
- ❖ An entity may be **concrete** such as a person or book or it may be **abstract** like a holiday or a concept.
- ❖ An **entity set** is a set of entities of same type that share the same properties or attributes. For example the set of all persons who are customers at a given bank can be defined as an entity set customer. The individual entities that constitute a set are said to be the **extension of**

the entity set. For example the individual customers that constitute a set are the extension of the entity set customer.

❖ Entity sets do not need to be disjoint. For example it is possible to define the entity set of all employees of a bank (employees) and the entity set of all customers of the bank.

2.4 ATTRIBUTES:-

- Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.
- There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- ✘ **Simple attribute** – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- ✘ **Composite attribute** – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- ✘ **Derived attribute** – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- ✘ **Single-value attribute** – Single-value attributes contain single value. For example – Social_Security_Number.
- ✘ **Multi-value attribute** – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

2.3 RELATIONSHIP SETS:-

- ❖ A relationship is an association among several entities.
- ❖ A relationship set is a set of relationships of the same type.
- ❖ The association between entity sets is referred to as participation that is the entity sets E1, E2... En participates in a relationship set R.
- ❖ A relationship instance in an E-R schema represents an association between the named entities of the real world enterprise that is being modelled.
- ❖ A relationship may also have attributes called **descriptive attributes**. Consider a relationship set depositor with entity sets customer and account. We could associate the attribute access date to that relationship to specify the most recent date on which the customer accessed an account.
- ❖ A relationship instance in a given relationship set must be uniquely identifiable from its participating entities, without using descriptive attributes. For example instead of using a single access date we use access dates.
- ❖ However there can be more than one relationship set involving the same entity set. For example guarantor in customer –loan relationship.

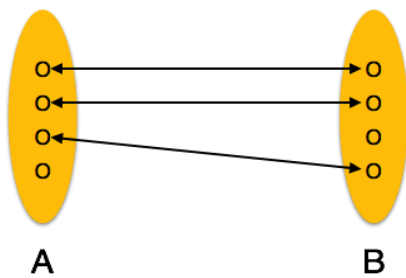
- ❖ One that involves two entity sets is called **binary relationship**. Most of the relationship sets in a database system are binary.
- ❖ The relationship set works on among employees, branch and job is an example of ternary relationship.
- ❖ The number of entity sets that participate in a relationship set is also called the **degree of relationship set**. A binary relationship set is of degree two and a ternary relationship set is of degree three.

2.5 MAPPING CONSTRAINTS:-

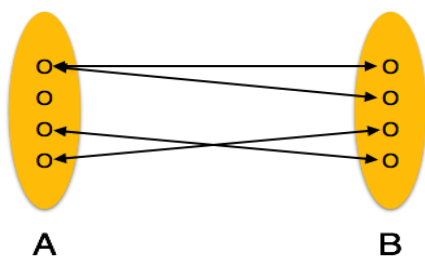
An E-R enterprise schema may define certain constraints to which the contents of the database must conform. Mapping cardinalities and participation constraints are two of the most important types of constraints.

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

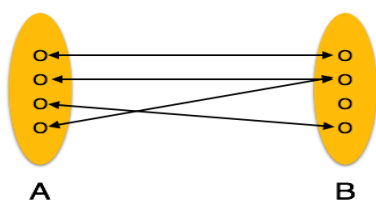
- **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



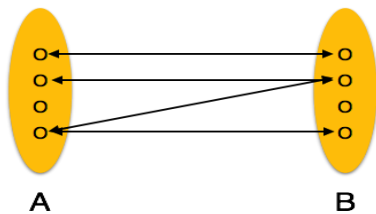
One-to-many – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



Many-to-one – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



Many-to-many – One entity from A can be associated with more than one entity from B and vice versa.



Participation constraints:-

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

For example we expect every loan entity to be related to at least one customer through the borrower relationship. Therefore the participation of the loan in the relationship set borrower is total. In contrast an individual can be a bank customer whether or not she has a loan with the bank. Hence it is possible that only some of the customer entities are related to the loan entity through the borrower relationship and hence the participation is partial.

2.7 RELATIONAL DATA MODEL:-

- ❖ This model has the advantage of being simple in principle; users can express their queries in a powerful query language.
- ❖ In this model the relation is the only construct required to represent the associations among the attributes of an entity as well as relationships among different entities.
- ❖ One of the main reasons for introducing this model was to increase the productivity of the application programmer by eliminating the need to change application programs when a change is made to the database. Users need not know the exact physical structures to use the database and are protected from any changes made to these structures. They are however still required to know how the data has been partitioned into various relations.
- ❖ The relation is the only data structure used in the relational data model to represent both entities and relationships between them. A relation may be visualized as a named table.
- ❖ Rows of the relation are referred to as **tuples** of the relation and the columns are its **attributes**. Each attribute of a relation has a distinct name. The values for an attribute or a column are drawn from a set of values known as **domain**.

Domain:- a Domain definition specifies the kind of data represented by the attribute.

Key of a relation

Primary key: - The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key;

Foreign key:- FOREIGN KEY is a column that creates a relationship between two tables. The purpose of foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

Difference between Primary key & Foreign key

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

2.8 NETWORK DATA MODEL:-

- ❖ The network data model was formalized in the late 1960's by the Database Task Group of Conference on data system languages (DBTG/CODASYL). Hence it is also known as DBTG model.
- ❖ The network model uses two different data structures to represent the database entities and relationship between the entities named *record type* and *set type*. A record type is used to represent an entity type. It is made up of a number of data items that represents the attributes of an entity.

A set type is used to represent a directed relationship between two record types, the so called owner record type and the member record type.

- ❖ The set type like the record type is named and specifies that there is a one to many relationship (1:M) between the owner and member record types. The set type can have more than one record type as its member, but only one record type is allowed to be the owner in a given set type.
- ❖ A database could have one or more occurrences of each of its record types and set types. An occurrence of a set type consists of an occurrence of the owner record type and any number of occurrences of each of its member record type. A record type can't be a member of two distinct occurrences of the same type.
- ❖ To avoid the confusion inherent in the use of the word 'set' to describe the mechanism for showing relationship in the network model, the other terms suggested are co set, fan set, owner coupled set, CODASYL set, DBTG set etc.
- ❖ Bachman introduced a graphical means called a data structure diagram to denote the logical relationship implied by the set. Here a labeled rectangle represents the corresponding entity or record type. An arrow that connects two labeled rectangles represents a set type. The arrow direction is from owner record type to member record type. In the network model, the relationships as well as the navigation through the database are predefined at database creation time.

2.9 HIERARCHICAL DATA MODEL:-

- ❖ A tree may be defined as a set of nodes such that there is one specially designated node called root node and the remaining nodes are partitioned into disjoint sets, each of which in turn is a tree, the sub trees of the root. If the relative order of the sub trees is significant the tree is an ordered tree.
- ❖ In a hierarchical database the data is organized in a hierarchical or ordered tree structure and the database is a collection of such disjoint trees (sometimes referred to as forests or spanning trees). The nodes of the tree represent record types. Each tree effectively represents a root record type and all its dependent record types. If we define the root record type at level 0, then the level of its dependent record types can be defined at level 1. The dependents of the record types at level 1 are said to be at level 2 and so on.
- ❖ An occurrence of a hierarchical tree type consists of one occurrence of the root record type along with zero or more occurrences of its dependent sub tree types. Each dependent sub tree is in turn, hierarchical and consists of a record type as its root node.
- ❖ In a hierarchical model no dependent record can occur without its parent record occurrence. Furthermore no dependent record occurrence may be connected to more than one parent record occurrence.
- ❖ A hierarchical model can represent a one to many relationships between two entities where the two are respectively parent and child. However to represent many to many relationship requires duplication of one of the record types corresponding to one of the entities involved in this relationship. Note that such duplications lead to inconsistencies when only one copy of a duplicate record is updated.

Model Questions

2.0 DATA MODELS

1. Define data Model. Explain various types of Data model. (10)
2. What is an Attribute? What are different types of attribute explain all. (6)
3. Explain the symbols used in ER diagram. What is the significance of E-R diagram? (6)
4. Define Primary key. (2)
5. What is a Foreign Key? (2)
6. What are the differences between hierarchical & network data model? (6)
7. Write the comparison of three data models. (10)
8. What is an Entity Sets? (2)
9. Explain Relationship sets with example. (6)
10. What is Attribute? What are various types of attributes used? (6)
11. What are symbol used to design ER diagram. (2)
12. What do you mean by data integrity? (2)
13. What is mapping constraints? (2)
14. Define Relational database. (2)
15. What is candidate key? (2)
16. What are the properties of Primary key? (2)
17. Define Tuple and attribute. (2)

CHAPTER-03

RELATIONAL DATABASE

QUERY LANGUAGE:-

- ❖ A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.
- ❖ Query languages can be categorized as either procedural or non-procedural. In a procedural language the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language the user describes the desired information without giving a specific procedure for obtaining that information.

3.1 RELATIONAL ALGEBRA:-

- ❖ The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as its operands and produces a new relation as its result.
- ❖ The fundamental operations in the relational algebra are select, project, union, set difference, rename and Cartesian product.
- ❖ In addition to the fundamental operations there are several other operations namely set intersection, natural join, division and assignment.

3.2 Fundamental Operations:-

The select, project and rename operations are called unary operations because they operate on one relation. The other three operations union, set difference and Cartesian product operate on pairs of relations and therefore called binary relations.

Select Operation:-

The select operation selects tuples that satisfy a given predicate. We use the lower case Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ . Thus to select those tuples of the loan relationship where the branch is "Bhubaneswar", we write

$\sigma_{\text{branch_name}=\text{Bhubaneswar}}$ (loan)

- In general we allow comparisons using $=, \neq, \leq, <, \geq, >$ in the selection predicate,
- Further we can combine several predicates into a larger predicate by using the connectors \wedge (AND), \vee (OR) and \sim (NOT). For example

$\sigma_{\text{branch_name}=\text{Bhubaneswar} \wedge \text{amount} > 1200}$

- The selection predicate may include comparisons between two attributes.

Project Operation:-

- The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relationship is a set, any duplicate rows are eliminated.

- Projection is denoted by upper case Greek letter Π (pi). Suppose we want to list all loan numbers and the amounts of loans but do not care about the branch name.

$$\Pi \text{ loan_number, amount (loan)}$$

Composition of relational operations:-

For example we want to find those customers who live in **Bhubaneswar**?

$$\Pi \text{ customer_name}(\sigma \text{ customer_city} = \text{ " Bhubaneswar" }(\text{customer}))$$

Here instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

- In general since the result of a relational algebra operation is of the type relation as its inputs, relational algebra operations can be composed together into a relational algebra expression.
- Composing relational algebra operations into relational algebra expressions is just like composing arithmetic operations i.e. Inner parentheses will be evaluated first and then the outer parentheses and so on.

Union operation:-

Consider a query to find the names of all bank customers who have either an account or a loan account or both.

The customer relation does not contain information about loan and the loan relation does not contain information about customer. So to find all the customer of a particular bank we have to join both relations by the following projection operation.

$$\Pi \text{ customer_name}(\text{depositor}) \cup \Pi \text{ borrower_name}(\text{loan})$$

To have union operation we require two conditions.

1. The relations R and S must be of same arity i.e. they must have same number of attributes.
2. The domains of the ith attribute of R and the ith attribute of S must be same for all i.

Set Difference operation:-

- The set difference operation is denoted by “-“ allows us to find tuples that are in one relation but not are in the other one. The expression R-S produces a relation containing those tuples that are in R but not in S
- We can find all customers of the bank who have an account but not a loan.

$$\Pi \text{ customer_name}(\text{deositor}) - \Pi \text{ customer_name}(\text{borrower})$$

We must ensure that set difference are taken between compatible relations. Therefore a set difference operation R-S to be valid we require that the relations R and S be of same arity and the domains of ith attribute of R and the ith attribute of S be same for all i.

Cartesian product operation:-

- The Cartesian product operation denoted by “X” allows us to combine information from any two relations. We write the Cartesian product of relations R1 and R2 as R1 X R2
- A relation is by definition a subset of a Cartesian product of a set of domains.
- However since the same attribute name may appear in both R1 and R2 we need to devise a naming schema to distinguish between these attributes.
- We do it by attaching to an attribute the name of the relation from which the attribute originally came.

For example the relational schema R= borrower X depositor is

(depositor.name, depositor.accno, borrower.name, borrower. Loan no, borrower.accno)

Set intersection operation:-

- We want to find out the customers who have both a loan and an account. Using set intersection operation we write

$$\Pi \text{ depositor_name}(\text{depositor}) \cap \Pi \text{ borrower_name}(\text{borrower})$$

- We can rewrite any relational algebra expression that uses set intersection operation with a pair of set difference operation.

$$R \cap S = R - (R - S)$$

Natural Join operation:-

- The natural join is a simple operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the join symbol \bowtie . The natural join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas and finally removes duplicate attributes.

Although the definition of natural join is complicated the operation is easy to apply. For example find the names of all customers who have a loan in the bank and find the amount of loan.

Π customer_name, amount (depositor \bowtie borrower)

Definition of natural join:-

Consider two relations r(R) and s(S). The natural join of r and s, denoted by r \bowtie s formally defined as follows.

$$r \bowtie s = \Pi R \cup S (\sigma_{r.A1=s.A1 \wedge r.A2=s.A2 \wedge \dots \wedge r.An=s.An})$$

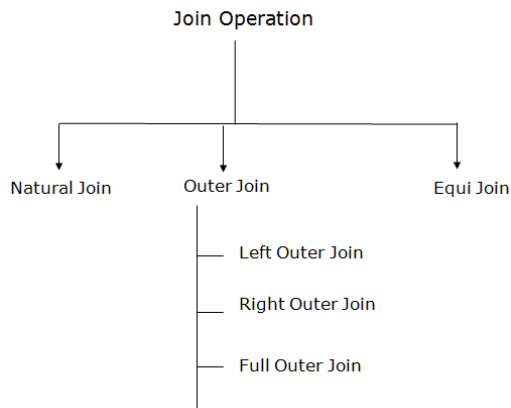
Where $R \cap S = \{A1, A2, A3, \dots, An\}$

- The natural join is associative i.e. if there are three relations P, Q and R, then associatively shows that (P Q) R = P (Q R)

What is Join?

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by \bowtie .

Types of Join operations:-



We can classify joins basically into two types

1. **INNER JOINS:** These joins are the one that has the tuples that satisfy some conditions and rest are discarded. Further they are classified as

- Theta join
- Equi join
- Natural join

2. **OUTER JOINS:** These have all the tuples from either or both the relations. Further they are classified as

- Left outer join
- Right outer join
- Full outer join

Let us now move on to the study the classified types with examples in detail.

INNER JOINS

1. **Theta join(θ)** – They have tuples from different relations if and only if they satisfy the theta condition, here the comparison operators (\leq , \geq , $<$, $>$, $=$, \neq) come into picture. Let us consider simple example to understand in a much better way, suppose we want to buy a mobile and a laptop, based on our budget we have thought of buying both such that mobile price should be less than that of laptop. Look at the tables below,

MOBILE

MODEL	PRICE
Asus	10k
Samsung	20k
Iphone	50k

LAPTOP

MODEL	PRICE
Acer	20k
HP	35k
Apple	80k

Now, we have considered the condition as the cost of the mobile should be less than that of laptop so our resulting table will have only those tuples that satisfy this condition.

AFTER JOINS

Asus	Acer
Asus	HP
Asus	Apple
Samsung	HP
Samsung	Apple
Iphone	Apple

2. Equi join – As the name itself indicates, if only equivalence conditions are used by theta join then it is called equi Join.

3. Natural join – It does not utilize any of the comparison operator. Here the condition is that the attributes should have same name and domain. There has to be at least one common attribute between between two relations. It forms the cartesian product of two arguments, performs selection forming equality on those attributes that appear in both relations and eliminates the duplicate attributes. Consider the example, where two tables namely employment table and department table have been shown.

EMPLOYMENT

NAME	EMPID	DPT_NAME
A	11	Sales
B	12	Finance
C	13	Finance

DEPARTMENT

DPT_NAME	MANAGER
Finance	M1
Sales	M2

Looking at above tables we realize that they have a common attribute called DPT_NAME, thus after the natural join the table becomes as

Name	EMPID	DPT_NAME	MANAGER
A	11	Sales	M2
B	12	Finance	M1
C	13	Finance	M1

Natural Join (\bowtie)

1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by \bowtie .

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. \bowtie EMP_NAME, SALARY (EMPLOYEE \bowtie SALARY)

Output:

EMP_NAME	SALARY
Amit	50000
Jaydeep	30000
Hari	25000

2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram Chandra	Civil line	Mumbai
Shyam Kumar	Park street	Kolkata
Ravi Kumar	M.G. Street	Delhi
Harihara	Nehru Nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram Chandra	Infosys	10000
Shyam Kumar	Wipro	20000

Kuber	HCL	30000
Harihara	TCS	50000

Input:

1. (EMPLOYEE \bowtie FACT_WORKERS)

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram Chandra	Civil line	Mumbai	Infosys	10000
Shyam Kumar	Park street	Kolkata	Wipro	20000
Harihara	Nehru Nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

- a. Left outer join
- b. Right outer join
- c. Full outer join

Left Outer Join($R \ltimes S$)

A. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by \ltimes .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE \ltimes FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram Chandra	Civil line	Mumbai	Infosys	10000
Shyam Kumar	Park street	Kolkata	Wipro	20000
Harihara	Nehru street	Hyderabad	TCS	50000

Ravi Kumar	M.G. Street	Delhi	NULL	NULL
------------	-------------	-------	------	------

Right Outer Join: (R \bowtie S)

B. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by \bowtie .

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input:

1. EMPLOYEE \bowtie FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram Chandra	Infosys	10000	Civil line	Mumbai
Shyam Kumar	Wipro	20000	Park street	Kolkata
Harihara	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

Full Outer Join: (R \bowtie S)

C. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by \bowtie .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE \bowtie FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram Chandra	Civil line	Mumbai	Infosys	10000

Shyam Kumar	Park street	Kolkata	Wipro	20000
Harihara	Nehru street	Hyderabad	TCS	50000
Ravi Kumar	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator (=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	Amit
2	Jaydeep
3	Hari

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input:

1. CUSTOMER \bowtie PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	Amit	1	Delhi
2	Jaydeep	2	Mumbai
3	Hari	3	Noida

MODEL QUESTIONS

3.0

RELATIONAL DATABASE

1. What is Join? (2)
2. Name different types of join. (2)
3. What do you mean by self join? (2)
4. What are the advantages of using Join Operator? (2)
5. What is the use of Project and Select Operator? (2)
6. Name two Unary operators and two Primary operators. (2)
7. What is Join? Explain different types of join with examples. (10)
8. Identify basic set theoretic operations in relational algebra. Give one example of each? (6)
9. Name types of relational operators. (2)
10. What is LOSSLESS Join ? (2)
11. Lossless join (Short Note) (4)

CHAPTER-04

NORMALIZATION IN RELATIONAL SYSTEM

Keys:-

- ❖ A **key** allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help to uniquely identify relationships and thus to distinguish relationships from each other.
- ❖ A **super key** is a set of one or more attributes that taken collectively allow us to identify uniquely an entity in the entity set. For example the customer_id attribute of the entity set customer and customer_name & customer_id attributes of the entity set customer.
- ❖ The concept of a super key is not sufficient for our purpose, since we can see that a super key may contain extra attributes. So we are interested in super keys for which no proper subset is a super key. Such minimal super keys are called **candidate keys**. It is possible that several distinct sets of attributes could serve as a candidate key. For example customer_name, customer_street
- ❖ We use the term **primary key** to denote a candidate key that is chosen by the database engineer as the principal means of identifying entities within an entity set.

Functional Dependency

- ⊙ Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2, ..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.
- ⊙ Functional dependency is represented by an arrow sign (\rightarrow) that is, $X \rightarrow Y$, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Property of Functional Dependency

Property was developed by William Armstrong in 1974 to reason about functional dependencies.

The property suggests rules that hold true if the following are satisfied:

- **Transitivity**
If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$ i.e. a transitive relation.
- **Reflexivity**
 $A \rightarrow B$, if B is a subset of A.
- **Augmentation**
The last rule suggests: $AC \rightarrow BC$, if $A \rightarrow B$

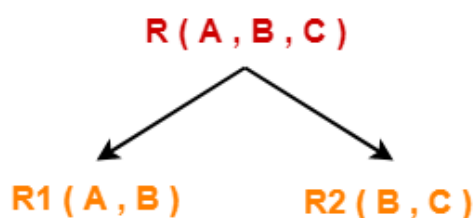
OR

If F is a set of functional dependencies then the closure of F , denoted as F^+ , is the set of all functional dependencies logically implied by F . Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- ⊙ **Reflexive rule** – If α is a set of attributes and β is subset of α , then α holds β .
- **Augmentation rule** – If $a \rightarrow b$ holds and y is attribute set, then $ay \rightarrow by$ also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule** – same as transitive rule in algebra, if $a \rightarrow b$ holds and $b \rightarrow c$ holds, then $a \rightarrow c$ also holds. $a \rightarrow b$ is called as a functionally that determines b .

4.2 LOSS LESS DECOMPOSITION:-

- When we decompose a relation into a number of smaller relations, it is crucial that the decomposition be lossless. We must first present criteria for determining whether a composition is lossy.
- Let R be relation schema and F be a set of functional dependencies on R . Let R_1 and R_2 form a decomposition of R . This composition is a loss less join decomposition of R if at least one of the following functional dependencies is in F^+ :
 - (i) $R_1 \cap R_2 \rightarrow R_1$
 - (ii) $R_1 \cap R_2 \rightarrow R_2$
- In other words if $R_1 \cap R_2$ forms a super key of either R_1 or R_2 the decomposition of R is a loss less decomposition.
- For the general case of decomposition of a relation into multiple parts at once the test for lossless join decomposition is more complicated.
- While the test for binary decomposition is clearly a sufficient condition for loss less join, it is a necessary condition only if all constraints are functional dependencies.



4.5 NORMALIZATION

What is Normalization?

- ⊙ **Normalization** is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updating anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

What is Dependency?

- ⊙ A dependency is a constraint that applies to or defines the relationship between attributes. It occurs in a database when information stored in the same database table

uniquely determines other information stored in the same table. You can also describe this as a relationship where knowing the values of one attribute (or a set of attributes) is enough to tell you the value of another attribute (or set of attributes) in the same table.

WHY WE NEED NORMALIZATION?

Normalization is the aim of well design Relational Database Management System (RDBMS). It is step by step set of rules by which data is put in its simplest forms. We normalize the relational database management system because of the following reasons:

- Minimize data redundancy i.e. no unnecessarily duplication of data.
- To make database structure flexible i.e. it should be possible to add new data values and rows without reorganizing the database structure.
- Data should be consistent throughout the database i.e. it should not suffer from following anomalies.
- Insert Anomaly - Due to lack of data i.e., all the data available for insertion such that null values in keys should be avoided. This kind of anomaly can seriously damage a database
- Update Anomaly - It is due to data redundancy i.e. multiple occurrences of same values in a column. This can lead to inefficiency.
- Deletion Anomaly - It leads to loss of data for rows that are not stored else where. It could result in loss of vital data.
- Complex queries required by the user should be easy to handle.
- On decomposition of a relation into smaller relations with fewer attributes on normalization the resulting relations whenever joined must result in the same relation without any extra rows. The join operations can be performed in any order. This is known as Lossless Join decomposition.
- The resulting relations (tables) obtained on normalization should possess the properties such as each row must be identified by a unique key, no repeating groups, homogenous columns, each column is assigned a unique name etc.

4.3 IMPORTANCE OF NORMALIZATION:-

- Searching, sorting, and creating indexes is faster, since tables are narrower, and more rows fit on a data page?
- You usually have more tables.
- You can have more clustered indexes (one per table), so you get more flexibility in tuning queries.
- Index searching is often faster, since indexes tend to be narrower and shorter.
- More tables allow better use of segments to control physical placement of data.
- You usually have fewer indexes per table, so data modification commands are faster.
- Fewer null values and less redundant data, making your database more compact.
- Triggers execute more quickly if you are not maintaining redundant data.
- Data modification anomalies are reduced.

- Normalization is conceptually cleaner and easier to maintain and change as your needs change.
- The cost of finding rows already in the data cache is extremely low.
- Avoids data modification (INSERT/DELETE/UPDATE) anomalies as each data item lives in one place.
- Fewer null values and less opportunity for inconsistency.
- A better handle on database security.
- Increased storage efficiency.
- The normalization process helps maximize the use of clustered indexes, which is the most powerful and useful type of index available. As more data is separated into multiple tables because of normalization, the more clustered indexes become available to help speed up data access.

ADVANTAGES OF NORMALIZATION

The following are the advantages of the normalization.

- More efficient data structure.
- Avoid redundant fields or columns.
- More flexible data structure i.e. we should be able to add new rows and data values easily
- Better understanding of data.
- Ensures that distinct tables exist when necessary.
- Easier to maintain data structure i.e. it is easy to perform operations and complex queries can be easily handled.
- Minimizes data duplication.
- Close modeling of real world entities, processes and their relationships.

DISADVANTAGES OF NORMALIZATION

The following are disadvantages of normalization.

- You cannot start building the database before you know what the user needs.
- On normalizing the relations to higher normal forms i.e. 4NF, 5NF the performance degrades.
- It is very time consuming and difficult process in normalizing relations of higher degree.
- Careless decomposition may leads to bad design of database which may leads to serious problems.

How many normal forms are there?

They are

- First Normal Form
- Second Normal Form
- Third Normal Form

- Boyce-Codd Normal Form
- Fourth Normal Form
- Fifth Normal Form

What do we mean when we say a table is not in normalized form?

Let's take an example to understand this,

Say I want to create a database which stores my friends name and their top three favorite artists.

This database would be quite a simple so initially I'll be having only one table in it say friends table. Here FID is the primary key.

Normalization of Database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant (useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

4.4 FIRST NORMAL FORM(1NF):-

- ❖ The first of the normal forms that we study, first normal form imposes a very basic requirement on relations; unlike the other normal forms, it does not require additional information such as functional dependency.
- ❖ A domain is atomic if elements of the domain are considered to be individual units. We say that a relation schema R is in first normal form (1NF) if the domains of all the attributes of R are atomic.
- ❖ In other words only one value is associated with each attribute and the value is not set of values or list of values.
- ❖ A database schema is in first normal form if every relation schema included in the database schema is in 1NF.
- ❖ The first normal form pertains to the tabular format of the relation.
- ❖ Sometimes non atomic values can be useful. For example composite valued attributes and set valued attributes. In many domains where entities have a complex structure, forcing a 1NF representation represents an unnecessary burden on the application programmer who has to write code to convert data into atomic form.

For example consider a table which is not in first normal form.

Student	Age	Subject
Asish	17	DBMS,OS

Piyush	14	OS
Kamal	15	OS

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows.

Student Table following 1NF will be:

Student	Age	Subject
Asish	17	DBMS,OS
Piyush	17	OS
Kamal	14	OS
Sagar	15	OS

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

SECOND NORMAL FORM (2NF):-

- ❖ A relation scheme $R\langle S, F \rangle$ is in 2NF if it is in 1NF and if all non prime attributes are fully functional dependent on the relation keys.
- ❖ A database schema is in 2NF if every relation schema included in the database schema is in 2NF
- ❖ A 2NF does not permit partial dependency between a non prime attribute and the relation keys.
- ❖ Even though 2NF does not permit partial dependency between a non prime attribute and the relation keys it does not rule out the possibility that a non prime attribute may also be functionally dependent on another non prime attribute. This type of dependency between non prime attributes also causes anomalies.

In example of First Normal Form there are two rows for Adam, to include multiple subjects that he has opted for. While this is searchable, and follows First normal form, it is an inefficient use of space. Also in the above Table in First Normal Form, while the candidate key is {**Student, Subject**}, **Age** of Student only depends on Student column, which is incorrect as per Second Normal Form. To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the student names as foreign keys.

New Student Table following 2NF will be :

Student	Subject
Asish	DBMS
Piyush	OS
Kamal	OS
Sagar	OS

In Subject Table the candidate key will be {**Student, Subject**} column. Now, both the above tables qualify for Second Normal Form and will never suffer from Update Anomalies. Although

there are a few complex cases in which table in Second Normal Form suffers Update Anomalies, and to handle those scenarios Third Normal Form is there

THIRD NORMAL FORM (3NF):-

❖ BCNF requires that all non trivial dependencies of the form $\alpha \rightarrow \beta$, where α is a super key. Third normal form (3NF) relaxes this constraint slightly by allowing certain non trivial functional dependencies whose left side is not a super key.

A relation schema R is in third normal form with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subset R$ and $\beta \subset R$, at least one of the following holds:

1. $\alpha \rightarrow \beta$ is a trivial functional dependency
2. α is a super key for R
3. Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

❖ Note that the third condition above does not say that a single candidate key must contain all the attributes in $\beta - \alpha$; each attribute A in $\beta - \alpha$ may be contained in a different candidate key.

❖ **Third Normal form** applies that every non-prime attribute of table must be dependent on primary key. The *transitive functional dependency* should be removed from the table. The table must be in **Second Normal form**.

For example, consider a table with following fields.

Student_id	Student_name	DOB	Street	City	State	PIN
------------	--------------	-----	--------	------	-------	-----

In this table Student_id is Primary key, but street, city and state depends upon PIN. The dependency between PIN and other fields is called **transitive dependency**. Hence to apply **3NF**, we need to move the street, city and state to new table, with **PIN** as primary key.

New Student_Detail Table :

Student_id	Student_name	DOB	PIN
------------	--------------	-----	-----

Address Table:

PIN	City	Street	State
-----	------	--------	-------

The advantage of removing transitive dependency is:-

- ❖ Amount of data duplication is reduced.
- ❖ Data integrity achieved.

BOYCE- CODD NORMAL FORM (BCNF):-

❖ Boyce- Codd normal form eliminates all redundancy that can be discovered based on functional dependencies.

❖ A relational schema R is in BCNF with respect to a set F of functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subset R$ and $\beta \subset R$, at least one of the following holds:

1. $\alpha \rightarrow \beta$ is a trivial functional dependency (that is $\beta \subset \alpha$)
2. α is a super key for schema R
- ❖ A database design is in BCNF if each member of the set of relation schemas that constitute the design is in BCNF.
- ❖ When we decompose a schema that is not in BCNF, it may be that one or more of the resulting schemas are not in BCNF. In such cases further decomposition is required, the eventual result of which is a set of BCNF schemas.
- ❖ The BCNF imposes a stronger constraint on the types of functional dependencies allowed in a relation. The only non trivial functional dependencies allowed in a relation. The only non trivial functional dependencies allowed in the BCNF are those functional dependencies whose determinants are candidate super keys of the relation.
- ❖ Any schema that satisfies BCNF also satisfies 3NF, since each of its functional dependencies would satisfy one of the first two alternatives. BCNF is therefore a more restrictive normal form than is 3NF.
- ❖ A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. In the above normalized tables in 3NF, Student_id is super-key in Student_Detail relation and PIN is super-key in Address relation. So,
 $Student_id \rightarrow Student_name, DOB, PIN$
 And
 $PIN \rightarrow Street, City, State$
 confirms, that both relations are in BCNF.

MODEL QUESTIONS

4.0 NORMALIZATION IN RELATIONAL SYSTEM

1. What is Functional Dependencies? Explain its properties. (6)
2. What are the advantages of normalization? (6)
3. What are objectives of Normalization? (6)
4. What is Normalization? (2)
5. Define full functional dependence. (2)
6. Define BCNF? (2)
7. Short note on BCNF. (4)
8. What is Normalization? What are the advantages of Normalization? (6)
9. What is Normalization? Explain 1NF, 2NF, 3NF with example of each? (10)
10. Explain need of normalization in DBMS. Explain 1NF, 2NF, 3NF? (10)
11. What is Lossless decomposition? Explain it. (6)

CHAPTER-05

STRUCTURED QUERY LANGUAGE

5.1 QUERY LANGUAGE:-

- ❖ A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.
- ❖ Query languages can be categorized as either procedural or non-procedural. In a procedural language the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language the user describes the desired information without giving a specific procedure for obtaining that information.

5.2 QUERIES IN SQL:-

Table:-

- ❖ A table is a database object that holds user data. The simplest analogy is to think of a table as a spread sheet
- ❖ The columns of a table are associated with a specific data type.
- ❖ Oracle ensures that only data which is identical to the data type of the column will be stored within the column.

What is SQL?

SQL is a programming language for Relational Databases. It is designed over relational algebra and tuple relational calculus. SQL comes as a package with all major distributions of RDBMS.

SQL comprises both data definition and data manipulation languages. Using the data definition properties of SQL, one can design and modify database schema, whereas data manipulation properties allows SQL to store and retrieve data from database.

Introduction to SQL

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases

- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
 - SQL can retrieve data from a database
 - SQL can insert records in a database
 - SQL can update records in a database
 - SQL can delete records from a database
 - SQL can create new databases
 - SQL can create new tables in a database
 - SQL can create stored procedures in a database
 - SQL can create views in a database
 - SQL can set permissions on tables, procedures, and view
-

SQL is a Standard:-

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

RDBMS:-

RDBMS stands for **Relational Database Management System**. RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Every table is broken up into smaller entities called **fields**.

The fields in the Customers table consist of Customer ID, Customer Name, Contact Name, Address, City, Postal Code and Country.

A **field** is a column in a table that is designed to maintain specific information about every record in the table.

A **record**, also called a row, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.

A column is a vertical entity in a table that contains all information associated with a specific field in a table

Data type:-

1. Character (size):-

This data type is used to store character string of fixed length. The size in bracket determines the number of characters the shell can hold. Maximum size is 255 characters.

2. Varchar(size)/ Varchar2(size):-

This data type is used to store variable length alpha numeric data. Maximum size is 2000 characters.

3. Date:-

Date data type is used to represent date and time. The standard format is DD-MMM-YY.

4. Number:-

The number data type is used to store numbers (fixed or floating) number of any magnitude may be stored up to 38 digit of decision. Maximum size is $9.99 * 10^{124}$. The precision P determines the maximum length of data where as the scale S determines the number of places to right of the decimal.

5. Long:-

This data type is used to store variable length character, strings up to 2GB. Long data can be used to store arrays of binary data in ASCII format.

6. RAW/ LONG RAW:-

The raw data type is used to store binary data such as picture or image. Raw data type can have maximum 255 byte and maximum size of long raw is up to 2GB.

5.3 SQL COMMANDS:-

Command to Create a Table:-

Syntax:-

Create table tablename (colname datatype(size), colname datatype (size),....., colname datatype (size));

Example:-

Create table student (rollno varchar2(20), name varchar2(30), address varchar2(50),semester varchar2(10));

To view the structure of a table:-

Syntax:-

Sql> desc table name;

Example:-

Desc student;

Column name	Data type	size
Rollno	varchar2	20
Name	varchar2	30
Address	varchar2	50
Semester	varchar2	10

Command to insert data into a table:-

While inserting a single row of data into a table, the insert operation first creates a new row (empty in the database table) and then loads the value passed into the column specified.

Syntax:-

Insert into table name (col1, col2, col3,....., coln) values (expr1, expr2, expr3,....., exprn);

Example:

Insert into student ('F18014024001','Radhika','BBSR','3RD CSE');

Insert into student ('F18014024002','Rupak','CTC','5TH CSE');

Insert into student ('F18014024003','Jyoti','RKL','5TH IT');

Insert into student ('F18014024004','Ajay','BAM','3RD IT');

Insert into student ('F18014024005','Manoj','KHD','5TH CSE');

Viewing data of a table:-

In order to view the total data of the table the syntax is as follows.

Syntax:-

Sql> Select * from table name;

Example:-

Select * from student;

Rollno	Name	Address	Semester
F18014024001	Radhika	BBSR	3RD CSE
F18014024002	Rupak	CTC	5TH CSE
F18014024003	Jyoti	RKL	5TH IT
F18014024004	Ajay	BAM	3RD IT
F18014024005	Manoj	KHD	5TH CSE

In order to view the partial column data we have the following syntax.

Syntax:-

Sql> Select col1, col2, col5 from table name;

Example:-

Select Rollno, Semester from student;

Rollno	Semester
F18014024001	3RD CSE
F18014024002	5TH CSE
F18014024003	5TH IT
F18014024004	3RD IT
F18014024005	5TH CSE

Filtering table data:-

There are three ways provided by the oracle to filter data.

i) Selected rows and all columns:-

Syntax:-

Sql> Select * from table name where <condition>;

Example:-

Select * from student where Semester='5TH CSE';

Rollno	Name	Address	Semester
F18014024002	Rupak	CTC	5TH CSE
F18014024005	Manoj	KHD	5TH CSE

ii) Selected columns and all rows:-

Syntax:-

Sql> select col1, col2, col4 from table name;

Example:-

Select Rollno, Name, Semester from student;

Rollno	Name	Semester
F18014024001	Radhika	3RD CSE
F18014024002	Rupak	5TH CSE
F18014024003	Jyoti	5TH IT
F18014024004	Ajay	3RD IT
F18014024005	Manoj	5TH CSE

iii) Selected columns and selected rows:-

Syntax:-

Sql> select col1,col2, col4 from table name where <condition>;

Example:-

select rollno,name,semester from student where semester='5TH CSE';

Rollno	Name	Semester
F18014024002	Rupak	5TH CSE
F18014024005	Manoj	5TH CSE

Eliminating duplicate rows using a select statement:-

Syntax:-

Sql> select distinct col1, col2, from table name;

A table could hold duplicate rows. In such case to view only unique rows the syntax is as follows.

Example:-

select distinct Semester from student;

Semester
3RD CSE
5TH CSE
5TH IT
3RD IT

Syntax:-

Sql> select distinct * from table name;

Example:-

select distinct * from student;

Rollno	Name	Address	Semester
F18014024001	Radhika	BBSR	3RD CSE
F18014024002	Rupak	CTC	5TH CSE
F18014024003	Jyoti	RKL	5TH IT
F18014024004	Ajay	BAM	3RD IT
F18014024005	Manoj	KHD	5TH CSE
F18014024001	Jyoti	RKL	5TH IT

Base table

Rollno	Name	Address	Semester
F18014024001	Radhika	BBSR	3RD CSE
F18014024002	Rupak	CTC	5TH CSE
F18014024003	Jyoti	RKL	5TH IT
F18014024004	Ajay	BAM	3RD IT
F18014024005	Manoj	KHD	5TH CSE

Resulting table

Sorting data in a table:-

- Oracle allows data from a table to the view in a sorted order.
- The rows retrieved from the table will be sorted in either ascending or descending order. In case if there is no mention of the sorting order, the oracle engine sorts the data in ascending order by default.

Syntax:-

Sql> select * from table name order by col1, col2 desc;

Example:-

Select * from student order by roll no desc;

Rollno	Name	Address	Semester
F18014024005	Manoj	KHD	5TH CSE
F18014024004	Ajay	BAM	3RD IT
F18014024003	Jyoti	RKL	5TH IT
F18014024002	Rupak	CTC	5TH CSE
F18014024001	Radhika	BBSR	3RD CSE

Inserting data into a table created from another table:-

Syntax:-

Sql> insert into new table name select col1, col2, col3, col4 from old table name where<condition>;

Example:-

Insert into course select Rollno, Name, Address, Semester from student where Name='Ajay';

Regdno	Sname	Address	Semester
F18014024004	Ajay	BAM	3RD IT

Deleting data from a table:-

❖ To remove all the rows from a table the syntax is as follows

Syntax:-

Sql> delete from table name;

Example:-

delete from student;

❖ To remove a set of rows from a table the syntax is as follows

Syntax:-

Sql> delete from table name where <condition>;

Example:-

delete from student where Name='Radhika';

Rollno	Name	Address	Semester
F18014024004	Ajay	Bam	3RD IT
F18014024003	Jyoti	Rkl	5TH IT
F18014024002	Rupak	Ctc	5TH CSE
F18014024005	Manoj	Khd	5TH CSE

Updating the contents of a table:-

The update command is used to change or modify data value of a table.

Syntax:-

Sql> update table name set col1=<expression>, col2=<expression>;

Example:-

Update student set address='bam';

To update records conditionally we can use the following syntax.

Syntax:-

Sql> update table name set col. name=<expression> where <condition>;

Example:-

Update student set name='bidya' where rollno=' F18014024008';

Rollno	Name	Address	Semester
F18014024004	Ajay	BAM	3RD IT
F18014024001	Radhika	BBSR	3RD CSE
F18014024003	Jyoti	RKL	5TH IT
F18014024002	Rupak	CTC	5TH CSE
F18014024008	Bidya	KHD	5TH CSE

Modifying the structure of a table:-

To add a new column:-

Syntax:-

Sql> alter table table name add (column1 (data type (size)), (column2 (data type (size))) ;

Example:-

Alter table student add (dob date);

To drop a column from a table:-

Syntax:-

Sql> alter table table name drop column name;

Example:-

Alter table student drop dob;

To modify columns from a table:-

Syntax:-

Sql> alter table table name modify (column name (new data type (new size)));

Example:-

Alter table student modify (name varchar2(50));

Renaming tables:-

To rename a table the syntax is as follows.

Syntax:-

Sql> rename old table name to new table name;

Example:-

Rename student to diploma_student;

To truncate a table:-

To truncate a table the syntax is as follows.

Syntax:-

Sql> truncate table table name;

Example:-

Truncate table student;

Destroying tables:-

To destroy a table with all its contents the syntax is as follows.

Syntax:-

Sql> drop table table name;

Example:-

Drop table student;

MODEL QUESTIONS

5.0 STRUCTURED QUERY LANGUAGE

1. Define DDL and DML. (2)
2. Write down the syntax & give examples for the following SQL commands: (10)
 - a. ALTER
 - b. INSERT
 - c. UPDATE
3. Explain with syntax different types of database languages used in SQL. (10)
4. What is the difference between DROP and DELETE command? (2)
5. What is the difference between CHARACTER and VARCHAR? (2)
6. What is the use of COMMIT and ROLLBACK Command? (2)
7. What is the difference between GRANT and REVOKE Command? (2)
8. Name data types used in SQL. (2)
9. What is the difference between NULL and NOTNULL? (2)
10. Write down the syntax & give examples for the following SQL commands: (10)

- a. INSERT
- b. UPDATE
- c. CREATE
- d. DELETE

11. What is SQL?

(2)

CHAPTER-06

TRANSACTION PROCESSING CONCEPTS

6.1 TRANSACTION PROCESSING:-

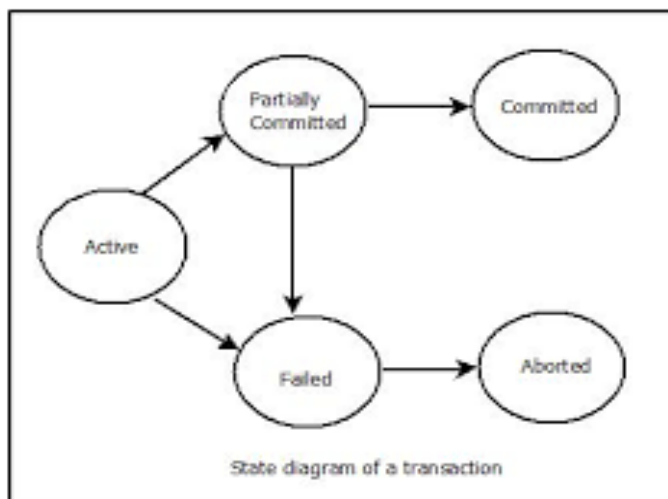
- ❖ A transaction is a program unit whose execution may change the contents of a database
- ❖ If the database was in a consistent state before a transaction, then on completion of the transaction the database will be in a consistent state.
- ❖ This requires that the transaction be considered atomic, it is executed successfully or in case of errors the user can view the transaction as not having been executed at all.

6.2 TRANSACTION CONCEPT:-

States of transaction:-

A transaction can be considered to be an atomic operation by the user but in reality it goes through a number of states during its life time.

States of Transactions



The various states of a Database Transaction are listed below

State	Transaction types
Active State	A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.
Partially Committed	A transaction goes into the partially committed state after the end of a transaction.
Committed State	When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.
Failed State	A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state.
Terminated State	State of transaction reaches terminated state when certain transactions which are leaving the system can't be restarted.

Let's study a state transition diagram that highlights how a transaction moves between these various states.

1. Once a transaction starts execution, it becomes active. It can issue READ or WRITE operation.
2. Once the READ and WRITE operations complete, the transactions becomes partially committed state.
3. Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the committed state.
4. If the check is a fail, the transaction goes to the Failed state.
5. If the transaction is aborted while it's in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.
6. The terminated state refers to the transaction leaving the system.

What are ACID Properties?

ACID PROPERTIES are used for maintaining the integrity of database during transaction processing. ACID stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability.

- **Atomicity:** A transaction is a single unit of operation. You either execute it entirely or do not execute it at all. There cannot be partial execution.

- **Consistency:** Once the transaction is executed, it should move from one consistent state to another.
- **Isolation:** Transaction should be executed in isolation from other transactions (no Locks). During concurrent transaction execution, intermediate transaction results from simultaneously executed transactions should not be made available to each other. (Level 0,1,2,3)
- **Durability:** After successful completion of a transaction, the changes in the database should persist. Even in the case of system failures.

What is a Schedule?

A Schedule is a process creating a single group of the multiple parallel transactions and executing them one by one. It should preserve the order in which the instructions appear in each transaction. If two transactions are executed at the same time, the result of one transaction may affect the output of other.

Example

```
Initial Product Quantity is 10
Transaction 1: Update Product Quantity to 50
Transaction 2: Read Product Quantity
```

If Transaction 2 is executed before Transaction 1, outdated information about the product quantity will be read. Hence, schedules are required.

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transaction are interleaved with some other transaction.

- **Schedule** – A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
- **Serial Schedule** – It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

Recoverable Schedule:

For each pair of transaction T_i and T_j , if T_j reads an object previously written by T_i , T_j commits after T_i commits

Avoids-cascading-abort Schedule:

For each pair of transaction T_i and T_j , if T_j reads an object previously written by T_i , T_i commits before the read operation of T_j .

Strict Schedule: An object written by T cannot be read or overwritten until T commits or aborts.

RECOVERABILITY:-**Crash Recovery**

Though we are living in highly technologically advanced era where hundreds of satellite monitor the earth and at every second billions of people are connected through information technology, failure is expected but not every time acceptable.

DBMS is highly complex system with hundreds of transactions being executed every second. Availability of DBMS depends on its complex architecture and underlying hardware or system software. If it fails or crashes amid transactions being executed, it is expected that the system would follow some sort of algorithm or techniques to recover from crashes or failures.

Failure Classification

To see where the problem has occurred we generalize the failure into various categories, as follows:

Transaction failure

When a transaction is failed to execute or it reaches a point after which it cannot be completed successfully it has to abort. This is called transaction failure. Where only few transaction or process are hurt.

Reason for transaction failure could be:

- **Logical errors:** where a transaction cannot complete because of it has some code error or any internal error condition
- **System errors:** where the database system itself terminates an active transaction because DBMS is not able to execute it or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability systems aborts an active transaction.

System crash

There are problems, which are external to the system, which may cause the system to stop abruptly and cause the system to crash. For example interruptions in power supply failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk failure:

In early days of technology evolution, it was a common problem where hard disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or part of disk storage Structure We have already described storage system here. In brief, the storage structure can be divided in various categories:

- **Volatile storage:** As name suggests, this storage does not survive system crashes and mostly placed much closer to CPU by embedding them onto the chipset itself for examples: main memory, cache memory. They are fast but can store a small amount of information.
- **Non-volatile storage:** These memories are made to survive system crashes. They are huge in data storage capacity but slower in accessibility. Examples may include, hard disks, magnetic tapes, flash memory, non-volatile (battery backed up) RAM.

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify data items. As we know that transactions are made of various operations, which are

Atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained that is, either all operations are executed or none.

When DBMS recovers from a crash it should maintain the following:

- It should check the states of all transactions, which were being executed.
- A transaction may be in the middle of some operation; DBMS must ensure the atomicity of transaction in this case.
- It should check whether the transaction can be completed now or needs to be rolled back.
- No transactions would be allowed to leave DBMS in inconsistent state.

There are two types of techniques, which can help DBMS in recovering as well as maintaining the atomicity of transaction:

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory and later the actual database is updated.

Log-Based Recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to actual modification and stored on a stable storage media, which is failsafe.

Log based recovery works as follows:

- The log file is kept on stable storage media
- When a transaction enters the system and starts execution, it writes a log about it <Tn, Start>
- When the transaction modifies an item X, it writes logs as <Tn, X, V1, V2>
- It reads Tn has changed the value of X, from V1 to V2.
- When transaction finishes, it logs: <Tn, commit>

Database can be modified using two approaches:

1. **Deferred database modification:** All logs are written on to the stable storage and database is updated when transaction commits.

2. Immediate database modification: Each log follows an actual database modification. That is, database is modified immediately after every operation.

Recovery with concurrent transactions

When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery it would become hard for recovery system to backtrack all logs, and then start recovering. To ease this situation most modern DBMS use the concept of 'checkpoints'.

MODEL QUESTIONS

6.0 TRANSACTION PROCESSING CONCEPTS

1. What is transaction processing? (2)
2. Name the process of transaction? (6)
3. What do you mean by ACID? (2)
4. Name the state of transaction? (6)
5. What is the scheduling of transaction? (2)
6. Define schedule?
(2)
7. What do you mean by recoverability schedule? (6)
8. What is meant by transaction? Explain with example? (6)
9. What is transaction? Explain different properties of transaction? (10)
10. Explain the serializable schedule? (6)
11. How we can test the serializability of the schedule? (6)
12. Explain different state of transaction with suitable diagram. (6)
13. What do you mean by schedule? Explain with suitable example. (6)

CHAPTER-07

CONCURRENCY CONTROL CONCEPTS

What is Concurrency Control?

- **Concurrency control** is the procedure in DBMS for managing simultaneous operations without conflicting with each another. Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical database, would have a mix of reading and write operations and hence the concurrency is a challenge.
- **Concurrency control** is used to address such conflicts which mostly occur with a multi-user system. It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases.
- Therefore, concurrency control is a most important element for the proper functioning of a system where two or multiple database transactions that require access to the same data, are executed simultaneously.

Need of Concurrency Control:

- 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.

Locks in DBMS:

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. Locks are used as a means of synchronizing the access by concurrent transactions to the database item.

Types of Lock:

Several types of locks are used in concurrency control. To introduce locking concepts gradually, we first discuss binary locks, which are simple but restrictive and so are not used in practice. We then discuss shared/exclusive locks, which provide more general locking capabilities and are used in practical database locking schemes.

- Binary Lock
- Live Lock
- Dead Lock

Binary Lock:

A binary lock can have two states or values:

Locked and Unlocked

- A distinct lock is associated with each database item A . If the value of the lock on A is 1, item A cannot be accessed by a database operation that requests the item. If the value of the lock on A is 0 then item can be accessed when requested. We refer to the current value of the lock associated with item A as $LOCK(A)$.
- There are two operations, lock item and unlock item are used with binary locking A transaction requests access to an item A by first issuing a lock *item (A)* operation. If $LOCK(A) = 1$, the transaction is forced to wait. If $LOCK(A) = 0$ it is set to 1 (the transaction locks the item) and the transaction is allowed to access item A . When the transaction is through using the item, it issues an unlock *item (A)* operation, which sets $LOCK(A)$ to 0 (unlocks the item) so that A may be accessed by other transactions. Hence binary lock enforces mutual exclusion on the data item.

Live Lock:

- A live lock is similar to a deadlock, except that the states of the processes involved in the live lock constantly change with regard to one another, none progressing. Live lock is a special case of resource starvation; the general definition only states that a specific process is not progressing.
- As a real-world example, live lock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they always both move the same way at the same time.
- Live lock is a risk with some algorithms that detect and recover from deadlock. If more than one process takes action, the deadlock detection algorithm can repeatedly trigger. This can be avoided by ensuring that only one process (chosen randomly or by priority) takes action.

Dead Lock:

- In a database, a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as it brings the whole system to a Halt.

- **Example** – let us understand the concept of Deadlock with an example : Suppose, Transaction T1 holds a lock on some rows in the Students table and **needs to update** some rows in the Grades table. Simultaneously, Transaction **T2 holds** locks on those very rows (Which T1 needs to update) in the Grades table **but needs** to update the rows in the Student table **held by Transaction T1**.

Now, the main problem arises. Transaction T1 will wait for transaction T2 to give up lock, and similarly transaction T2 will wait for transaction T1 to give up lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.

How Should Lock be used:-

In a transaction, a data item which we want to read/write should first be locked before the read/write is done. After the operation is over, the transaction should then unlock the data item so that other transaction can lock that same data item for their respective usage. For example we have a transaction to deposit Rs 100/- from account A to account B. The transaction should now be written as:

Lock-X (A); (Exclusive Lock, we want to both read A's value and modify it)

Read A;

A = A - 100;

Write A;

Unlock (A);

(Unlocking A after the modification is done)

Lock-X (B); (Exclusive Lock, we want to both read B's value and modify it)

Read B;

B = B + 100; Write B; Unlock (B); (Unlocking B after the modification is done)

And the transaction that deposits 10% amount of account A to account C should now be written as:

Lock-S (A); (Shared Lock, we only want to read A's value)

Read A;

Temp = A * 0.1; Unlock (A);

(Unlocking A)

Lock-X (C); (Exclusive Lock, we want to both read C's value and modify it)

Read C; C = C + Temp; Write C; Unlock (C); (Unlocking C after the modification is done)

Now let us see how these locking mechanisms help us to create error free schedules. You should remember that in the previous chapter we discussed an example of an erroneous schedule:

<p>T1</p> <p>Read A;</p> <p>A = A - 100;</p> <p>Write A;</p> <p>Read B;</p> <p>B = B + 100;</p> <p>Write B</p>	<p>T2</p> <p>Read A;</p> <p>Temp = A * 0.1;</p> <p>Read C;</p> <p>C = C + Temp;</p> <p>Write C;</p>
---	--

We detected the error based on common sense only that the Context Switching is being

performed before the new value has been updated in A. T2 reads the old value of A, and thus deposits a wrong amount in C. Had we used the locking mechanism, this error could never have occurred. Let us rewrite the schedule using the locks.

<p>T1</p> <p>Lock-X (A)</p> <p>Read A;</p> <p>A = A - 100;</p> <p>Write A;</p> <p>Write A;</p> <p>Unlock (A)</p> <p>Lock-X (B)</p> <p>Read B;</p> <p>B = B + 100;</p> <p>Write B;</p> <p>Unlock (B)</p>	<p>T2</p> <p>Lock-S (A)</p> <p>Read A;</p> <p>Temp = A * 0.1;</p> <p>Unlock (A)</p> <p>Lock-X(C)</p> <p>Read C;</p> <p>C = C + Temp;</p> <p>Write C;</p> <p>Unlock (C)</p>
--	---

We cannot prepare a schedule like the above even if we like, provided that we use the locks in the transactions. See the first statement in T2 that attempts to acquire a lock on A. This would be impossible because T1 has not released the exclusive lock on A, and T2 just cannot get the shared lock it wants on A. It must wait until the exclusive lock on A is released by T1, and can begin its execution only after that. So the proper schedule would look like the following:

<p>T1</p> <p>Lock-X (A)</p> <p>Read A;</p> <p>A = A - 100;</p> <p>Write A;</p> <p>Unlock (A)</p> <p>Lock-X (B)</p> <p>Read B;</p>	<p>T2</p> <p>Lock-S (A)</p> <p>Read A;</p> <p>Temp = A * 0.1;</p> <p>Unlock (A)</p> <p>Lock-X(C)</p> <p>Read C;</p> <p>C = C + Temp;</p> <p>Write C;</p> <p>Unlock (C)</p>
--	---

```
B = B + 100;  
Write B;  
Unlock (B)
```

And this automatically becomes a very correct schedule. We need not apply any manual effort to detect or correct the errors that may creep into the schedule if locks are not used in them.

Avoiding database deadlocks:-

Application developers can eliminate all risk of enqueue deadlocks by ensuring that transactions requiring multiple resources always lock them in the same order.

First it highlights the fact that processes must be inside a transaction for deadlocks to happen. Note that some database systems can be configured to cascade deletes which creates an implicit transaction which then can cause deadlocks. Also some DBMS vendors offer row-level locking a type of record locking which greatly reduces the chance of deadlocks as opposed to page level locking which creates many times more locks.

Second, by "multiple resources" this means more than one row in one or more tables. An example of locking in the same order would be to process all INSERTS first, all UPDATES second, and all DELETES last and within processing each of these handle all parent table changes before children table changes; and process table changes in the same order such as alphabetically or ordered by an ID or account number.

Third, eliminating all risk of deadlocks is difficult to achieve as the DBMS has automatic lock escalation features that raise row level locks into page locks which can be escalated to table locks. Although the risk or chance of experiencing a deadlock will not go to zero as deadlocks tend to happen more on large, high-volume, complex systems, it can be greatly reduced and when required the software can be enhanced to retry transactions when a deadlock is detected.

Fourth, deadlocks can result in data loss if the software is not developed to use transactions on every interaction with a DBMS and the data loss is difficult to locate and creates unexpected errors and problems.

7.3 SERIALIZABILITY:-

Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order. It assumes that all accesses to the database are done using read and write operations.

- **Serializability** is a property of a transaction schedule (history). It relates to the *isolation* property of a database transaction.
- **Serializability** of a schedule means equivalence to a *serial schedule* (i.e., sequential with no transaction overlap in time) with the same transactions. It is the major criterion for the correctness of concurrent transactions' schedule, and thus supported in all general purpose database systems.
- **The rationale behind serializability** is the following:

- If each transaction is correct by itself i.e. meets certain integrity conditions, then a schedule that comprises any *serial* execution of these transactions is correct (its transactions still meet their conditions): "Serial" means that transactions do not overlap in time and cannot interfere with each other, i.e. complete *isolation* between each other exists. Any order of the transactions is legitimate, if no dependencies among them exist, which is assumed. As a result, a schedule that comprises any execution (not necessarily serial) that is equivalent to any serial execution of these transactions is correct.
- Schedules that are not serializable are likely to generate erroneous outcomes. Examples are with transactions that debit and credit accounts with money: If the related schedules are not serializable, then the total sum of money may not be preserved. Money could disappear, or be generated from nowhere. It does not happen if serializability is maintained.
- If any specific order between some transactions is requested by an application, then it is enforced independently of the underlying serializability mechanisms. These mechanisms are typically indifferent to any specific order, and generate some unpredictable partial order that is typically compatible with multiple serial orders of these transactions.
- Two major types of serializability exist: *view-serializability*, and *conflict-serializability*. View-serializability matches the general definition of serializability given above. Conflict-serializability is a broad special case, i.e., any schedule that is conflict-serializable is also view-serializable, but not necessarily the opposite. Conflict-serializability is widely utilized because it is easier to determine and covers a substantial portion of the view-serializable schedules.
- **View-serializability** of a schedule is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that respective transactions in the two schedules read and write the same data values ("view" the same data values).
- **Conflict-serializability** is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that both schedules have the same sets of respective chronologically ordered pairs of conflicting operations (same precedence relations of respective conflicting operations).

MODEL QUESTIONS

7.0 CONCURRENTLY CONTROL CONCEPTS

1. What is lock? (2)
2. Name different type of lock? (2)
3. Distinguish between live lock & dead lock? (2)
4. What is serializability? (2)
5. What are disadvantages of binary lock? (2)
6. What is meant by lock? What are its types? (2)
7. What is lock? Explain different types of lock? (6)
8. Explain live lock & dead lock in a data base? (6)
9. Explain with example how serializability can obtain in concurrency control? (6)
10. How to handle dead locks and explain methods for dealing with the deadlock problem. (10)
11. Define concurring control. Explain the concepts live lock & dead lock? (10)

CHAPTER-08

SECURITY AND INTEGRITY

8.1 AUTHORIZATION:-

Authorization is the process where the database manager gets information about the authenticated user. Part of that information is determining which database operations the user can perform and which data objects a user can access.

OR

Authorization is the culmination of the administrative policies of the organization, expressed as a set of rules that can be used to determine which user has what type of access of which portion of database. The person who is in charge of specifying the authorization is usually called the authorizer. The authorizer is distinct from DBA and usually the person who owns the data.

❖ The authorization is usually maintained in the form of a table called an **access matrix**. The access matrix contains rows called **subject** and columns called **objects**. The entry in the matrix at the position corresponding to the intersection of a row and column indicate the **type of access** that the subject has with respect to the object.

Object:-

- An object is something that needs protection and one of the first steps in the authorization process is to select the objects to be used for security enforcement. Example: - a unit of data, views etc.
- The objects in the access matrix represent content independent access control. However to enforce content dependent access control, some structure for conditions or access predicates are incorporated in the access matrix.

Views as objects:-

Views or sub schemes can be used to enforce security. A user is allowed to access only that portion of the database defined by the user's view. A number of users may share a view. However the user may create new views based on the views allowed. The advantage of this approach is that the number of objects accessible to a class of users and the entry for it in the authorization table is reduced to one per view. This reduces the size of authorization matrix. The disadvantage is that the entire classes of users have the same access rights.

Granularity:-

This is used for security enforcement. This could be a file, a record or a data item. The smaller the protected object, the finer the degree of specifying protection. However the finer granularity increases the size of the authorization matrix and overhead in enforcing security.

Subject:-

A subject is an active element in the security mechanism. It operates on objects. A subject is a user who is given some rights to access a data object. We can also treat a class of users or an application program as a subject.

Access Types:-

The access allowed to a user could be for data manipulation or control. The manipulation operations are read, insert, delete, and update. The control operations are add, drop, alter and propagate.

- **Read:** Allows reading only the object.

- **Insert:** Allows inserting new occurrences of the object type. Insert access type requires that the subject has the read access as well. However it may not allow the modification of the existing data.
- **Delete:** Allows deleting an existing occurrence of the object type.
- **Update:** Allows the subject to change the value of the occurrence of the object. An update authorization may not include a delete authorization as well.

IEWS:-

- ✘ ❖ Sometimes for security and other concerns, it is undesirable to have all users to see the entire relation. It would also be beneficial if we would create useful relations for different groups of users, rather than have them all manipulate the base relations. Any relation that is not part of the physical database, i.e., a virtual relation is made available to the users is known as a **view**.
- ✘ It is possible to create views in SQL. A relation view is virtual since no corresponding physical relation exists. A view represents a different perspective of a base relation or relations.
- ✘ The result of a query operation on one or more base relations is a relation. Therefore if a user needs a particular view based on the base relations, it can be defined using a query expression. To be useful, we assign the view a name and relate it to the query expression.
- ✘ Create view <view name> as <query expression>
- ✘ A view is a relation (virtual rather than base) and can be used in query an expression that is queries can be written using views as a relation.
- ✘ Views generally are not stored, since the data in the base relations may change.
- ✘ The definition of a view in a create view statement is stored in the system catalog. Having been defined, it can be used as if the view really represents a real relation. However such a virtual relation defined by a view is recomputed whenever a query refers to it.
- ✘ Views or sub schemes are used to enforce security. A user is allowed access to only that portion of the database defined by the user's view.
- ✘ A number of users may share a view. However, the users may create new views based on the views allowed.
- ✘ The advantage of this approach is that the number of objects accessible to a class of users and the entry for it in the authorization matrix is reduced one per view. This reduces the size of authorization matrix. The disadvantage is that the entire class of users has the same access rights.

We can customize all aspects of a view, including:

- The name of the view
- The fields that appear in the view
- The column title for each field in the view
- The order of the fields in the view
- The width of columns in the view, as well as the overall width of the view
- The set of records that appear in the view (Filtering)
- The order in which records are displayed in the view (Sorting & Grouping)

- Column totals for numeric and currency fields (Totaling & Subtotaling)

8.2 SECURITY CONSTRAINTS:-

- ❖ Security in a database involves both policies and mechanisms to protect the data and ensure that it is not accessed, altered or deleted without proper authorization.
- ❖ There are four levels of defense or security constraints are generally recognized for database security: human factors, physical security, administrative control, and security and integrity mechanisms built into operating system and DBMS.

Human Factors:-

- At the outermost level are the human factors, which encompass the ethical, legal and social environments. An organization depends on these to provide a certain degree of protection. Thus it is unethical for a person to obtain something by stealth and it is illegal to forcibly enter the premises of an organization and hence the computing facility containing the database.
- Many countries have enacted legislation that makes it a crime to obtain unauthorized dial in access into computing system of an organization. Privacy laws also make it illegal to use information for purposes other than that for which it was collected.
- An organization usually performs some type of clearance procedure for personnel who are going to be dealing with sensitive information, including that contained in a database. This clearance procedure can be a very informal one, in the form of the reliability and trust that an employee has earned in the eyes of management or the clearance procedure could be a formal one.
- The authorizer is responsible for granting proper database access authorization to the user community. Assignment of authorization to a wrong class of users can result in possibly security violations.

Physical Security:-

- Physical security mechanisms include appropriate locks and keys and entry logs to computing facility and terminals.
- Security and physical storage devices (magnetic disk packs etc.) within the organization and when being transmitted from one location to another must be maintained. Access to the computing facility must be guarded, since an unauthorized person can make copies of files by bypassing the normal security mechanism built into the DBMS and the operating system.
- Authorized terminals from which database access is allowed to have to be physically secure, otherwise unauthorized persons may be able to glean information from the database using these terminals.

- User identification and passwords have to be kept confidential; otherwise unauthorized users can borrow the identification and password of a more privileged user and compromise the database.

Administrative Controls:-

- Administrative controls are the security and access control policies that determine what information will be accessible to what class of users and the type of access that will be allowed to this class.

DBMS and Operating System Mechanisms:-

- The proper mechanisms for the identification and verification of users. Each user is assigned an account number and a password. The operating system ensures that access to the system is denied unless the number and password are valid. In addition to the DBMS could also require a number and password before allowing the user to perform any database operations .
- The protection of data and programs, both in primary and secondary memories. This is usually done by the operating system to avoid direct access to the data in primary memory or to online files.
 - ❖ The DBMS has the following features for providing security and integrity mechanisms to support concurrency, transaction management, audit and recovery data logging. In addition the DBMS provides mechanisms for defining the authorization of the user community and specifying semantic integrity constraints and checking.

8.4 CRYPTOGRAPHY and ENCRYPTION:-

- ❖ Consider the secure transmission of this message:

“Mr. Watson, can you please come here”

One method of transmitting this message is to substitute a different character of the alphabet for each character in the message. If we ignore the space between words and the punctuation, and if the substitution is made by shifting each character by a different random amount then the above message can be transformed into, e.g. the following string of characters:

“xhlkunsikevoabondwinhwojahf”

- ❖ Cryptography has been practiced since the days of the Roman Empire. With the increasing use of public communication facilities to transmit data there is an increased need to make such transmissions secure. In a distributed environment, transmitting highly confidential information between geographically dispersed sites, in spite of the most stringent local security enforcement could lead to leakage.
- ❖ This points to the need for the data to be encrypted before it is transmitted. At the receiving end, the received data is deciphered before it is used. The sender must know how to encrypt the data and the receiver must know how to decipher the coded message. Since the computers at both ends can be used to cipher and decipher the data, the code used for ciphering is quite complex.

❖ The simple ciphering method used since the time of Julius Caesar called Caesar code. The one time code is a Caesar code used only once, which makes it difficult for the interceptor of the coded message to break the code since he or she does not have the opportunity to intercept more than one sample of the coded message, apply the distribution characteristics of the language and break the code.

❖ The other advantage of the onetime code is that breaking the code of a single transmission is not very helpful in deciphering subsequent coded messages, since each message will use a different code for encryption.

❖ However the drawback is that there must be an initial transmittal of the code that is to be used to the recipient, and for absolute unbreakability, the code has to be as long as the message that is transmitted.

A enciphering scheme developed by the U.S National Bureau of Standards (NBS) is called the **Data Encryption Standard (DES)**. This scheme is based on the substitution of characters and rearrangement of their order and assumes the existence of secure encryption keys. It is a relatively easy means to both encipher and decipher data. The algorithm is very well known and publicized but the encryption key is kept secret, which makes it very difficult for anyone who does not know the key to decipher the message. However the drawback in this scheme is that the encryption key has to be transmitted to the recipient before a message can be transmitted. Due to this drawback an encryption technique called one-way or trapdoor functions having the following characteristics:-

8.3 INTEGRITY CONSTRAINT:-

Integrity constraints ensure that any properly authorized access, alteration, deletion or insertion of the data in the database does not change the consistency and validity of the data. Database integrity involves the correctness of data; this correctness has to be preserved in the presence of concurrent operations, errors in the users operations and application programmes and failures in hardware and software. Constraints are restrictions or rules applied to a database to maintain its integrity.

They are-

1. Data/Entity integrity constraint

2. Referential integrity constraint

1. Data constraint:-

It is the most common integrity constraint also known as domain integrity constraint. Domain integrity rules are simply the definition of the domains of the attributes or the value set for the data items. The value that each attribute or data item can be assigned is expressed in the form of data type, a range of values or a value from a specified set. Example: In the relation EMPLOYEE the domain of the attribute Salary may be in the range of 12000 to 300000

The domain values supplied for an operation are validated against the domain constraint. In specifying the domain constraints, null values may or may not be allowed. Thus it is usual not to allow null values for any attribute that forms part of a primary key of a relation.

2. Referential integrity constraint:-

It is an implicit integrity constraint which states that, a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. The referential integrity rule is explained by the foreign key between two relation schemas R1 and R2.

A set of attributes Fk in relation schema R1 is a foreign key of R1 that references relation R2 if the following two rules are being satisfied.

i) The attributes in foreign key Fk have same domain as primary key attributes Pk of R2, the attributes Fk are said to be referenced or referred relation R2.

ii) A value of Fk in tuple T1 of the current state R1 (r1) either occurs as a value of Pk for some tuple T2 in the current state R2 (r2) or is none.

In the former case we have $T1 [Fk] = T2 [Pk]$ and we say that the tuple T1 references or refers to the tuple T2. R1 is called the referencing relation and R2 is called referenced relation.

MODEL QUESTIONS

8.0 SECURITY AND INTEGRITY

1. Define Authorization? (2)
2. What is data base security? (2)
3. Define data base integrity? (2)
4. Define View? (2)
5. What is Encryption? (2)
6. What is DES? (2)
7. What are disadvantages of encryption? (6)
8. Explain the importance of security in data base environment? (6)
9. What are various data security requirements? (6)
10. What are several of security level used to protect a data base? (6)
11. What do you mean by Authorization? Explain different types of Authorization? (10)
12. Define Data Encryption? Explain techniques used for encryption? (10)
13. What do you mean by DES? Explain it? (6)
14. Explain different algorithm used for encryption process? (6)
15. What is integrity constrains? Explain different level of constraints in a data base? (10)
16. Explain different types of constrains implemented in the tuples to ensure data integrity? (6)
17. What is the use of GRANT and REVOKE command? (2)
18. Name types of privileges used in database. (2)

REFERENCE BOOKS

1. Simple Approach to DBMS By Prateek Bhatia and Gurvinder Singh
 2. An Introduction to Database Systems By: - C.J. Date
 3. DATABASE System Concepts A. Silberschatz, H.F. Korth,
- .