Unit-1

1

INTRODUCTION TO DBMS

Unit Structure

1.0 OBJECTIVES

After going through this unit, you will able to:

- To introduce the concept of the DBMS with respect to the relational model.
- Define database, DBMS, overview of DBMS, level of abstraction, DBMS architecture.
- Learning Data models and its different types.
- Designing the database schema with the use of appropriate data types for storage of data in database.
- To create, manipulate, query and back up the databases.

1.1 INTRODUCTION

A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results. Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data.

DATABASE MANAGEMENT SYSTEM

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database.

1.2 OVERVIEW

The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a diskette, using a personal computer and software such as DBASE IV or V, Microsoft ACCESS, or EXCEL. A datum – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by information. Hence, information is interpreted data – data supplied with semantics. Knowledge refers to the practical use of information. While information can be transported, stored or shared without many difficulties the same cannot be said about knowledge. Knowledge necessarily involves a personal experience. Referring back to the scientific experiment, a third person reading the results will have information about it, while the person who conducted the experiment personally will have knowledge about it. The DBMS is a general purpose software system that facilitates the process of defining constructing and manipulating databases for various applications.

1.3 ADVANTAGES

Data Independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

Efficient Data Access: A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

Data Integrity and Security: If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.

Concurrent Access and Crash Recovery: A database system allows several users to access the database concurrently. Answering different questions from different users with the same (base) data is a central aspect of an information system. Such concurrent use of data increases the economy of a system. An example for concurrent use is the travel database of a bigger travel agency. The employees of different branches can access the database concurrently and book journeys for their clients. Each travel agent sees on his interface if there are still seats available for a specific journey or if it is already fully booked. A DBMS also protects data from failures such as power failures and crashes etc. by the recovery schemes such as backup mechanisms and log files etc.

Data Administration:

When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals, who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and fine-tuning the storage of the data to make retrieval efficient.

Reduced Application Development Time:

DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed from scratch because many important tasks are handled by the DBMS instead of being implemented by the application.

1.4 TYPES OF USERS IN DBMS:

Database Administrator:

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a **database administrator** (**DBA**). DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.

Naive users:

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a bank teller who needs to transfer $50 from account *A* to account *B* invokes a program called *transfer*. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

Application programmers:

Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports without writing a program.

Sophisticated users:

Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **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.

Specialized users:

Specialized users are sophisticated userswho write specialized database applications that do not fit into the traditional data-processing framework.

1.5 LEVELS OF ABSTRACTION IN A DBMS:

Hiding certain details of how the data are stored and maintained. A major purpose of database system is to provide users with an "Abstract View" of the data. In DBMS there are 3 levels of data abstraction. The goal of the abstraction in the DBMS is to separate the users request and the physical storage of data in the database.

Physical Level:

- The lowest Level of Abstraction describes "How" the data are actually stored.
- The physical level describes complex low level data structures in detail.

Logical Level:

- This level of data Abstraction describes "What" data are to be stored in the database and what relationships exist among those data.
- Database Administrators use the logical level of abstraction.

View Level:

- It is the highest level of data Abstracts that describes only part of entire database.

- Different users require different types of data elements from each database.
- The system may provide many views for the some database.

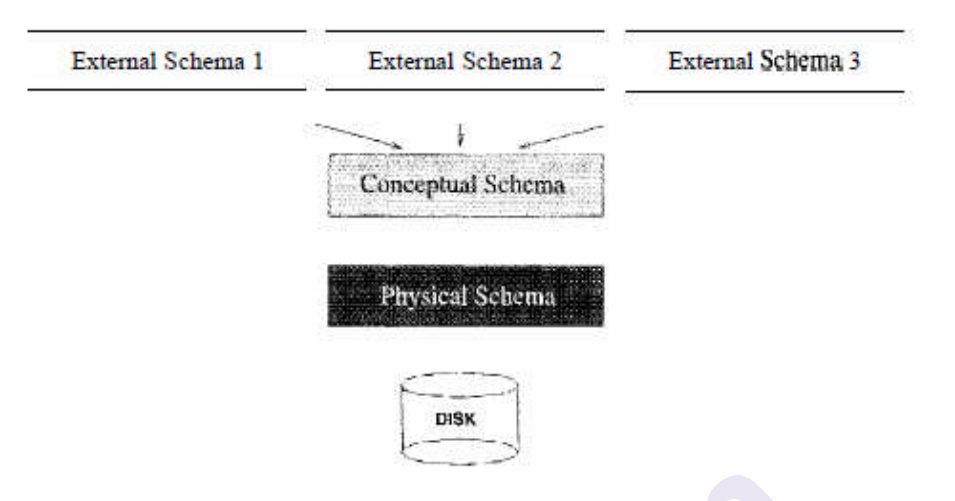


Figure 1.1: Level of Abstraction

1.6 DATA INDEPENDENCE:

A very important advantage of using DBMS is that it offers Data Independence. The ability to modify a scheme definition in one level without affecting a scheme definition in a higher level is called **data independence**.

There are two types:
1. Physical Data Independence
2. Logical Data Independence

Physical Data Independence:

- The ability to modify the physical schema without causing application programs to be rewritten.
- Modifications at this level are usually to improve performance.

Logical Data Independence:

- The ability to modify the conceptual schema without causing application programs to be rewritten
- Usually done when logical structure of database is altered
- Logical data independence is harder to achieve as the application programs are usually heavily dependent on the logical structure of the data.

1.7 DBMS ARCHITECTURE

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

The storage manager is important because databases typically require a large amount of storage space. Some Big organizations Database ranges from Giga bytes to Terabytes. So the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed. The query processor also very important because it helps the database system simplify and facilitate access to data. So quick processing of updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedural language.

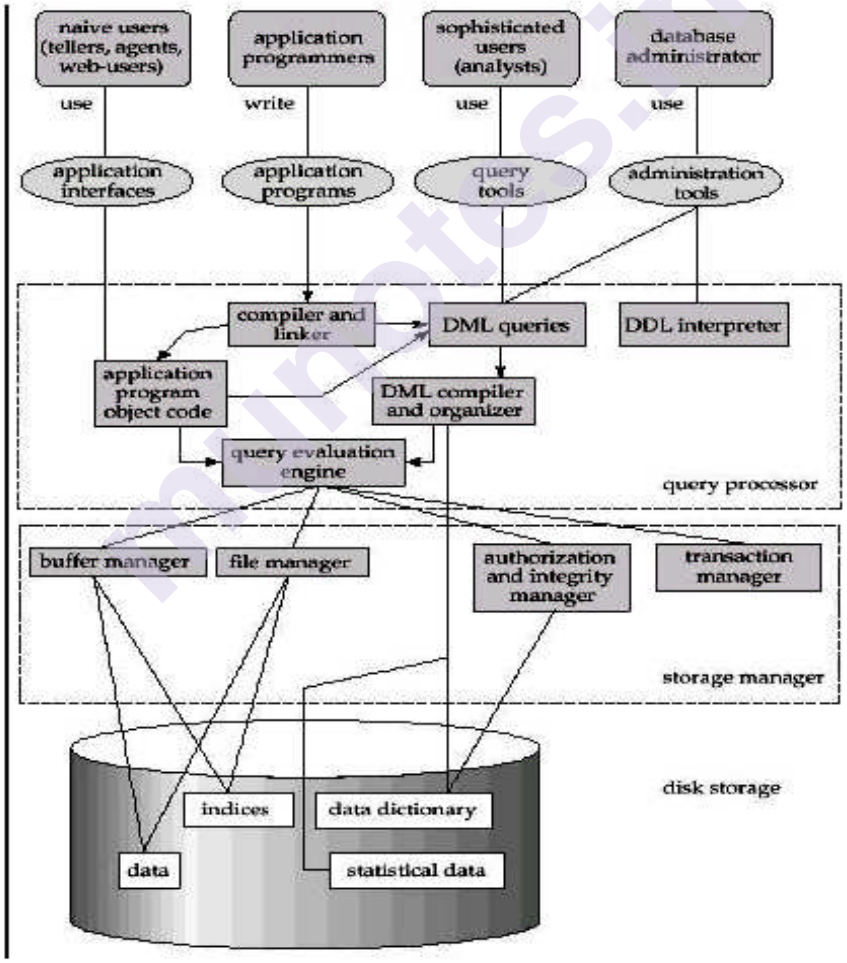


Figure 1.2: Database Architecture

Storage Manager:

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

Storage Manager Components:

Authorization and integrity manager: It tests for the satisfaction of integrity constraints and checks the authority of users to access data.

Transaction manager which ensures that the database itself remains in a consistent state despite system failures, and that concurrent transaction executions proceed without conflicting.

File manager: which manages the allocation of space on disk storage and the data structures used to representing information stored on disk.

Buffer manager: It is responsible for fetching data from disk storage into main memory. Storage manager implements several data structures as part of the physical system implementation. Data files are used to store the database itself. Data dictionary is used to stores metadata about the structure of the database, in particular the schema of the database.

Query Processor Components:

DDL interpreter: It interprets DDL statements and records the definitions in the data dictionary.

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

Query evaluation engine: It executes low-level instructions generated by the DML compiler.

Application Architectures:

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between client machines, on which remote database users' work, and server machines, on which the database system runs.

2

DATA MODELS

Unit Structure

2.0 Introduction

2.1 Types of Data Models

2.0 INTRODUCTION

Data models define how the logical structure of a database is modelled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system. The very first data model could be flat data-models, where all the data used are to be kept in the same plane. Earlier data models were not so scientific; hence they were prone to introduce lots of duplication and update anomalies. The following models are

2.1 TYPES OF DATA MODELS:

An Object Based Logical Model also known as conceptual data model which provides representation according to the way many users perceive data. Most popular conceptual data model is Entity Relationship Model which is based on the concepts of entity, relationship and attributes. Object based logical models provide flexible structuring capabilities and allow data constraints to be specified explicitly.

2.1.1 Entity Relationship Model

The ER (Entity relationship) based on the collection of basic objects, called entities and relationships among these objects. The diagrammatic notation associated with ER model, are also known as ER diagrams. The ER model employs three basic concepts entity sets relationship sets and attributes. An entity is an object in the real world that is distinguishable from all other objects. An entity set is a set of entities of the same type that share the same properties or attributes. Attributes are descriptive properties possessed by all members of an entity set.

2.1.2 Object Oriented Model

The object-oriented data model is an adaptation of the object-oriented programming language paradigm to database systems. The model is based on the concept of encapsulating data and code that operates on that data in an object. Entities in the sense of the ER model are

represented as objects with attributes values represented by instance variables within the object. The values stored in an instance variable are itself an object. Thus, a containment relationship i.e., is-part-of relationship is established among objects.

2.1.3 Physical data model

This model provides details of how data is stored on the computer storage media and meant for software specialist. This model hides many details of data storage on disk but can be implemented on a computer system directly. I is used in traditional commercial DBMS and based on the concepts of record structure with fixed format; hence it is also known as record based data model. The use of fixed length records simply the physical implementation of the database. The relational model is a primary data model in commercial data processing application.

2.1.4 Relational data model

This model uses a collection of tables to represent both data and the relationship among data. Tables are known as relations in relational database. Each relation consists of multiple columns and each column has unique name. This table has one column for each domain and one row for each tuple. Each column has a unique name which is called as attribute of the relation. The set of attributes are called as relation schema.

2.1.5 Network data model

The network model allows more general connections among the nodes. Network model has the ability to handle many –to – much relationship. The network data model is an abstraction of the design concepts used in the implementation of database.

2.1.6 Hierarchical data model

Data is sorted hierarchically in a tree like structure using parent child relationship, either in top down or bottom-up approach. This model uses pointers to navigate between stored data using hierarchical tree. Based on one-to-many relation.

2.1.7 Client server architecture:

The client/server architecture was developed to deal with computing environment in which a large number of PCs, workstations, file servers, printers, data base servers, Web servers, e-mail servers, and other software and equipment are connected via a network. The idea is to define specialized servers with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a file server that maintains the files of the client machines. Another machine can be designated as a printer server by being connected to various printers; all print requests by the clients are forwarded to this machine. Webservers or e-mail servers also fall into the specialized server category. The resources provided by specialized servers can be accessed by many client machines. The clientmachines provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local

applications. This concept can be carried over to other software packages, with specialized programs such as a CAD(computer-aided design) package being stored on specific server machines and being made accessible to multiple clients. Some machines would be client sites only (for example, diskless workstations or workstations or PCs with disks that have only client software installed).

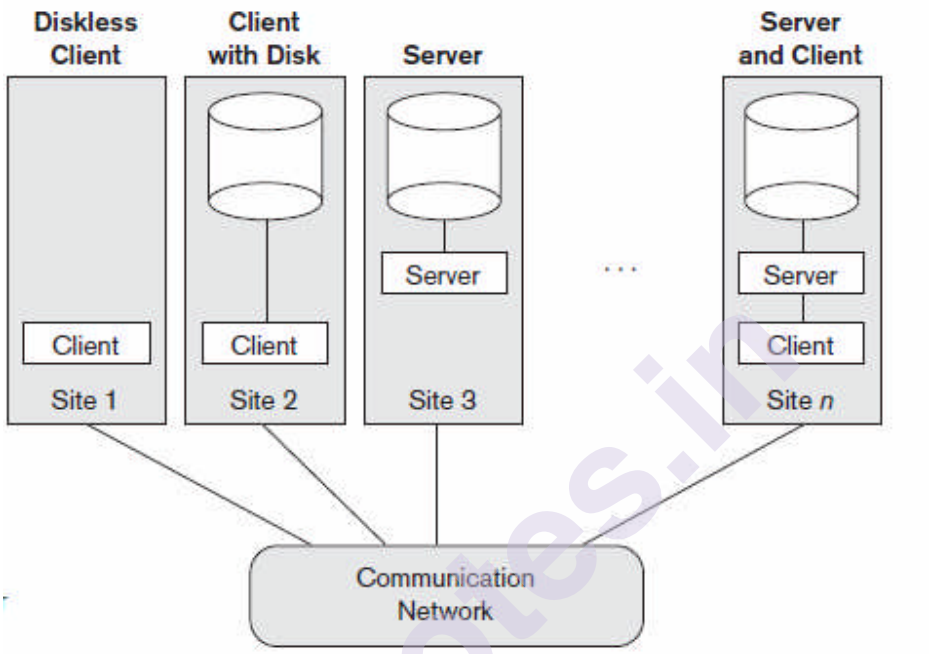


Figure 2.1: Client/Server architecture

3

ENTITY RELATIONSHIP MODEL

Unit Structure

3.0 INTRODUCTION

Entity-Relationship Model or simply ER Model is a high-level data model diagram. In this model, we represent the real-world problem in the pictorial form to make it easy for the stakeholders to understand. It is also very easy for the developers to understand the system by just looking at the ER diagram. We use the ER diagram as a visual tool to represent an ER Model. ER diagram has the following three components:

- **Entities**: Entity is a real-world thing. It can be a person, place, or even a concept. Example: Teachers, Students, Course, Building, Department,etc are some of the entities of a School Management System.
- **Attributes**: An entity contains a real-world property called attribute. This is the characteristics of that attribute. Example: The entity teacher has the property like teacher id, salary, age, etc.
- **Relationship**: Relationship tells how two attributes are related. Example: Teacher works for a department**.**
- **Relationship set:** A relationship set is a set of relationships of the same type. Formally it is a mathematical relation on (possibly non-distinct) sets. If are entity sets, then a relationship set R is a subset of Where is a relationship. For example, consider the two entity sets customer and account.
- **Key Constraints:** All the values of primary key must be unique. The value of primary key must not be null.
- **Participation Constraints:** We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints)
- **Weak entities:** In a relational database, a weak entity is an entity that cannot be uniquely identified by its attributes alone; therefore, it must use a foreign key in conjunction with its attributes to create a primary key.

- **Aggregation:** In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher-level entity.

3.1 TYPES OF ATTRIBUTES:

1. Simple Attributes

Simple attributes are atomic attributes with independence meaning which cannot be further divided. For example, employee's phone is an atomic attribute.

2. Composite Attributes:

Composite are made up of more than one attributes. It can divide into smaller subparts, which represent more basic attributes with independent meanings. They sometimes form a hierarchy. The value of a composite attribute is the combination of the values of its components atomics attributes. For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State and Zip. Composite attributes are useful to model situations in which a user sometimes refers to the composite attribute as a unit but at other times refers specifically to its components.

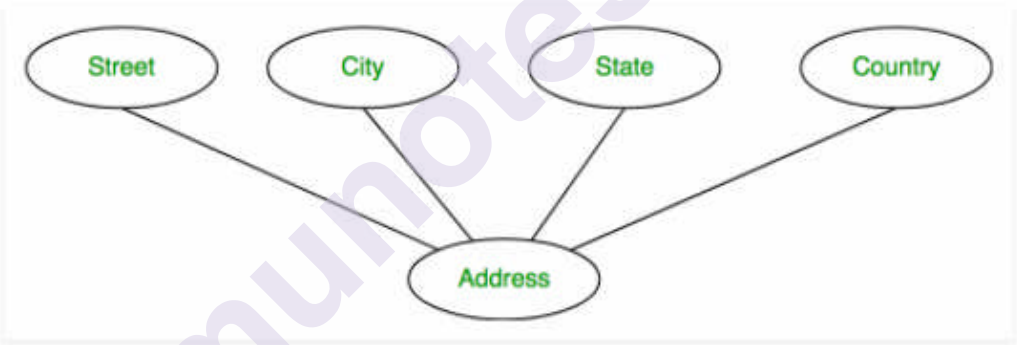


Figure 3.1: Composite attribute

Single valued attributes

Single valued attributes consist of individual or single value for a particular entity. For example, Employee id attributes refers to only one employee ID. Age attribute for a person. There may be instances where an attribute has a set of values for a specific entity. Suppose we add to the instructor entity set.

Multivalued Attributes:

Multi valued attributes has a group of values for a specific entity. Multi valued attributes comes with upper and lower limits the number of values to be specified for an entity. For example, an employee may have more than one phone number.

Stored Attributes: Stored attributes consist of attributes that are fetched directly from the entity. For Date_of_Birth

Derived Attributes: Data that is derived using the data stored in the stored attributes set are known as Derived attributes. For example, Age can be calculated using the stored date_of _birth attribute.

Entity type: an entity type defines a collection or set of entities that have the same attributes. Each entity type in the database is described by its name and attributes. For example, a college may want to store similar information concerning each of the students. Students can be entity types that share the same attributes, but each entity has its own values for each attribute.

Entity Set: The collection of all entities of a particular entity type in the database at any point in time is called an entity set. The entity set is usually referred to using the same name as the entity type. For example student refers to both a type of entity as well as the current set of all student entities in the database.

Relationship: An association among several entities is known as Relationship.

Figure 3.2: Relationship

Relationship set:

A relationship set is a set of relationship of the same type. It is a mathematical relation on n>=2 entity sets. Diamonds represents the relationship sets. If E1,E2,E3…….En are entity set then a relationship set R is a subset of {(e1,e2….en) | e1 ∈E1, e2 ∈E2…..en∈En where (e1,e2,……en) is a relationship.

Figure 3.3: Relationship set

Degree of relationship type

The degree of a relationship type is the number of participating entity types. A relationship type of degree two is called binary which are the most common one. A relationship type of degree three is called ternary. Higher degree relationships are more complex. Relationship in

databases is often binary. Some relationships that appear to be non-binary could actually be better represented by several binary relationships.

For example, one could create a ternary relationship parent, relating a child to his mother and father, such a relationship could also be represented by two binary relationships, mother and father relating a child to his mother and father separately. Using the two relationships mother and father provides us a record of a child's mother, even if we are not aware of the father's identity; a null value would be required if the ternary relationship parent is used. Using binary relationship sets is preferable in this case. For simplicity purpose it is always possible to replace a non-binary (n-ary, for n>2) relationship set by a number of distinct binary relationship sets.

Mapping Cardinality

The relationship set are of one to one, one to many, many to one or many to many. To distinguish among these types, either a directed line (→) or an undirected line (—)between the relationship set and the entity set is drawn.

- **One to one**:

  A directed line is drawn from the relationship set advisor to both entity sets instructor and student. This indicates that an instructor may advise at most one student, and a student may have at most one advisor.

- **One to many:**

  A directed line is drawn from the relationship set advisor to the entry set instructor and an undirected line to the entity set student. This indicates that an instructor may advise many students, but a student may have at most one advisor.

- **Many to one:**

  An undirected line drawn from the relationship set advisor to the entity set instructor and a directed line to the entity set student. This indicates that an instructor may advise at most one student, but a student may have many advisors.

- **Many to many:**

  an undirected line drawn from the relationship set advisor to both entity sets instructor and student. This indicates that an instructor may advise many students, and a student may have many advisors.

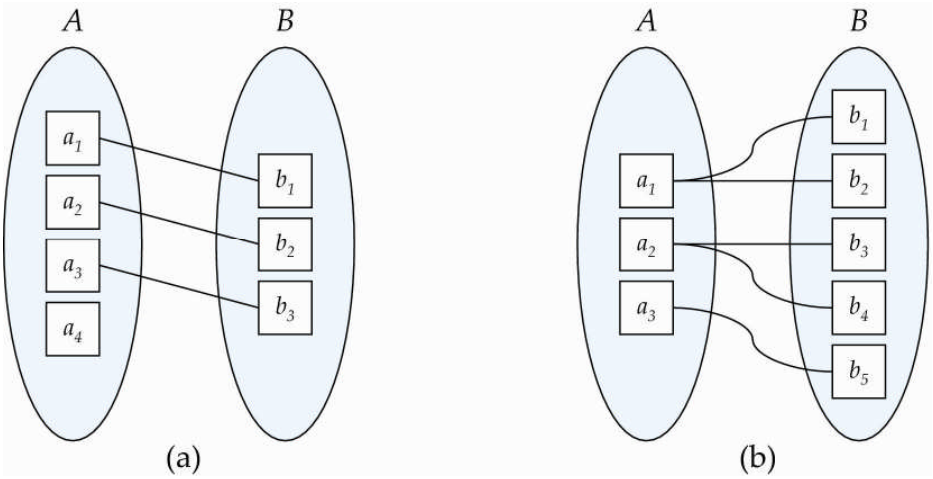
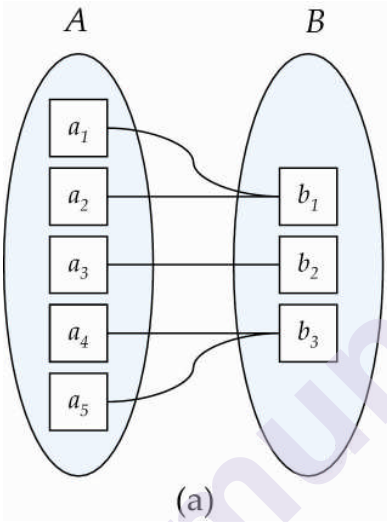
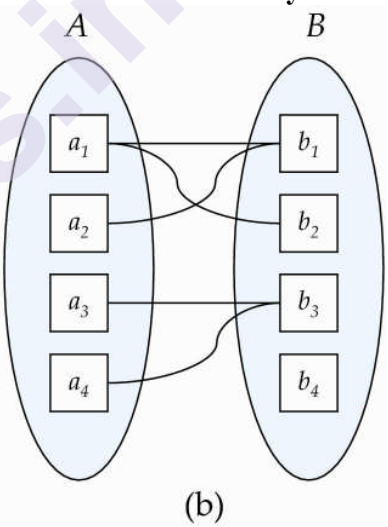


Figure 3.4: mapping cardinalities

One to One



One to many



Many to one

Many to many

Key constraint:

A Key or uniqueness constraint on the attributes of entities helps to identify relationship uniquely, and thus distinguish relationship from each other. No two entities are allowed to have exactly the same.

Specialization

Specialization is a process of creating sub parts of an entity type. Generalization is a bottom-up approach, while Specialization is a top-down approach. One higher level entity can be broken down into two lower-level entities by specialisation. The term "specialization" refers to a subset of an entity set that shares certain common characteristics. Normally, the superclass is described first, followed by the subclass and its related attributes, and finally the relationship set. For example, In an

employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.

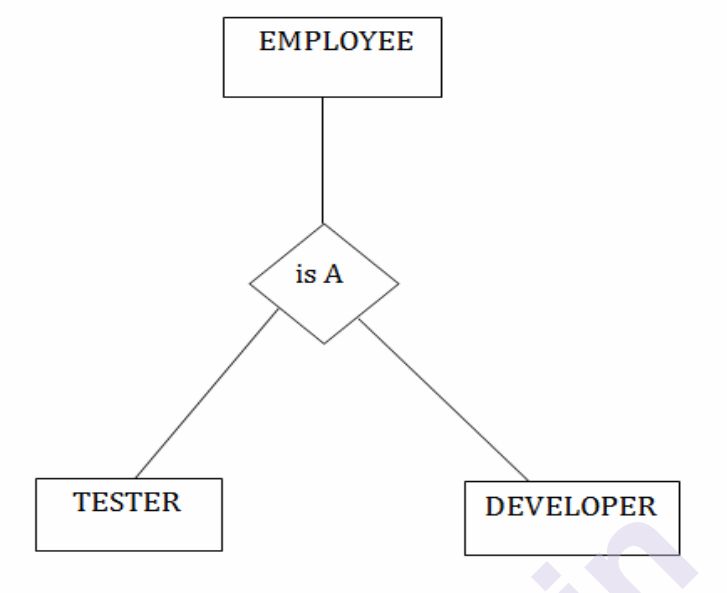


Figure 3.5: Specialization

Generalization

It's a reverse process of abstraction, where in the difference amongst the entity sets are suppressed and they are generalized together into a single entity type. Distinctions are made explicitly in case of generalization with top-down approach. Commonality is defined using generalization and expressed using containment relationship. It creates a relationship between higher-level entities set to successive hierarch of subclass entity set. The design process may also proceed in a bottom-up manner, in which multiple entities sets are synthesized into a higher-level entity set on the basis of common features. For example, Faculty and Student entities can be generalized and create a higher-level entity Person.

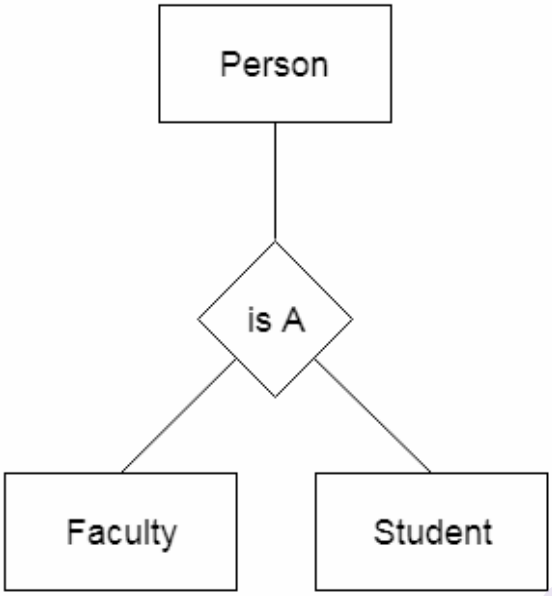


Figure 3.6: **Generalization**

AGGREGATION

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher-level entity. For example, Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

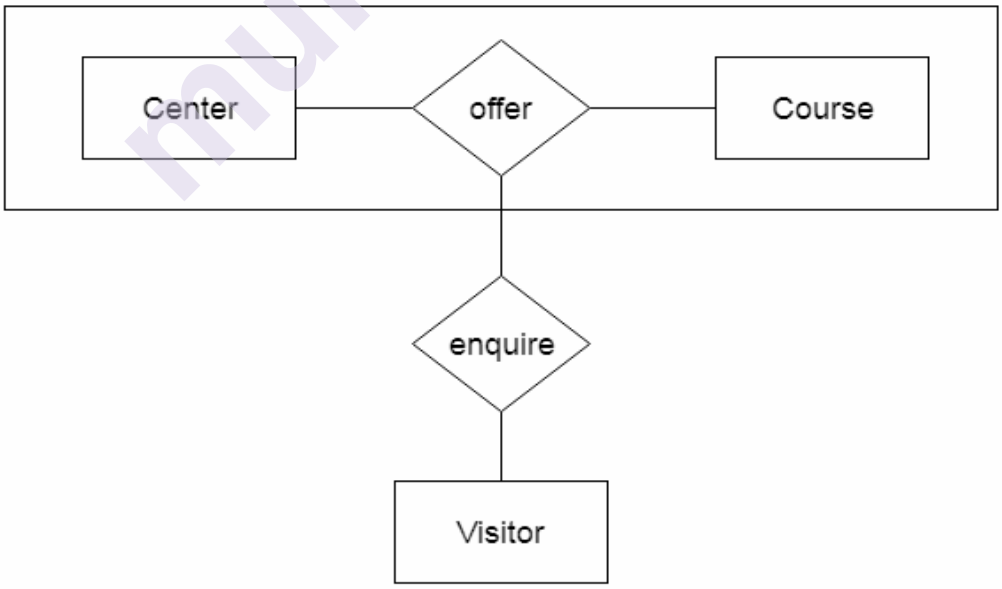


Figure 3.7: Aggregation

Summary of Notation in ER diagram

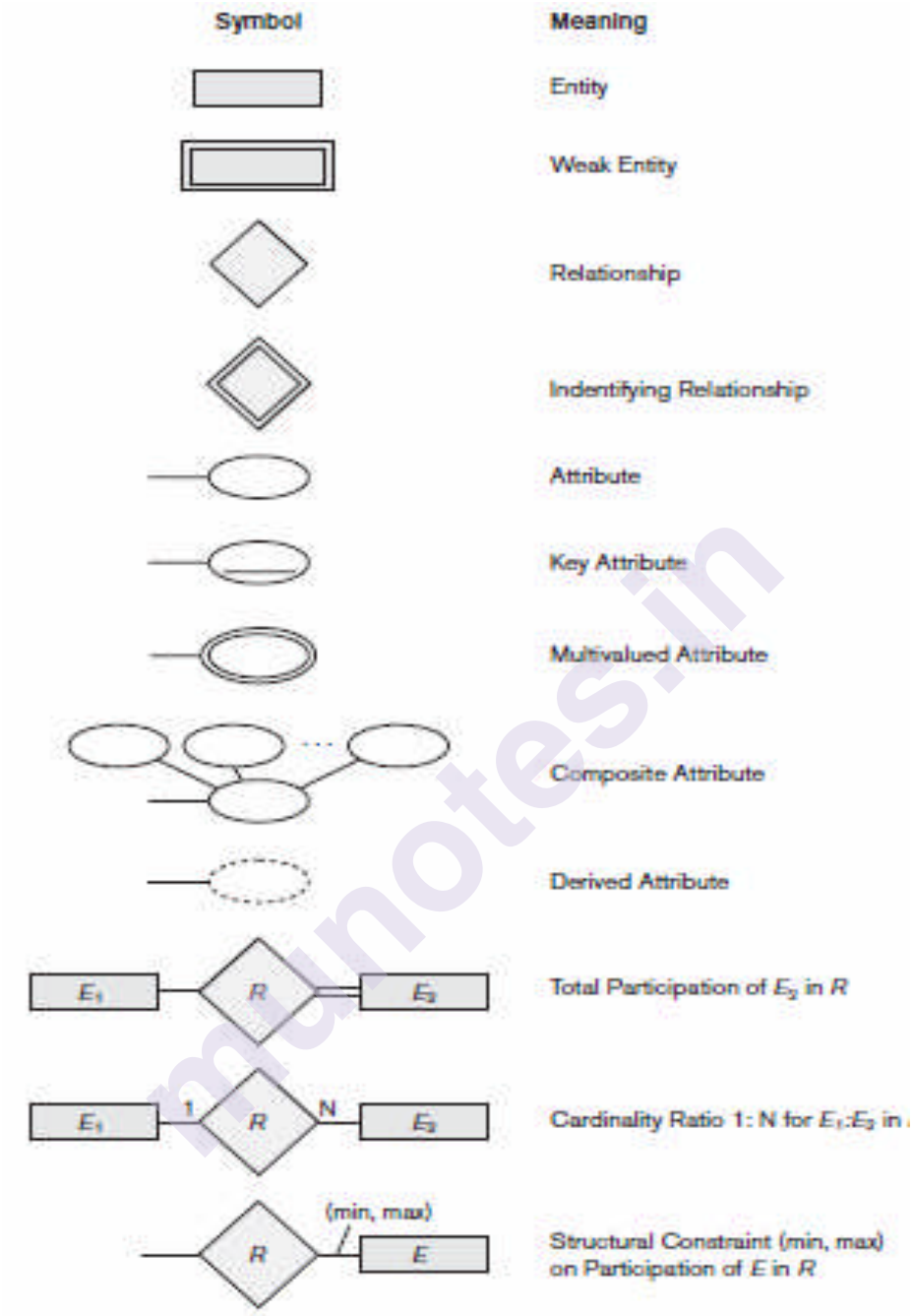


Figure3.8: summary of ER notation

Entity Vs Attributes

While identifying the attributes of an entity set, it is sometimes not clear whether a property should be modelled as an attribute or as an entity set (and related to the first entity set using a relationship set). For example, consider adding address information to the Employees entity set. One option is to use an attribute address. This option is appropriate if we need to record only one address per employee, and it suffices to think of an

address as a string. An alternative is to create an entity set called Addresses and to record associations between employees and addresses using a relationship.

Entity vs Relationship

The nature of ER modelling can thus make it difficult to recognize underlying entities, and we might associate attributes with relationships rather than the appropriate entities. In general, such mistakes lead to redundant storage of the same information and can cause many problems.

❖❖❖❖

4

RELATIONAL DATA MODEL

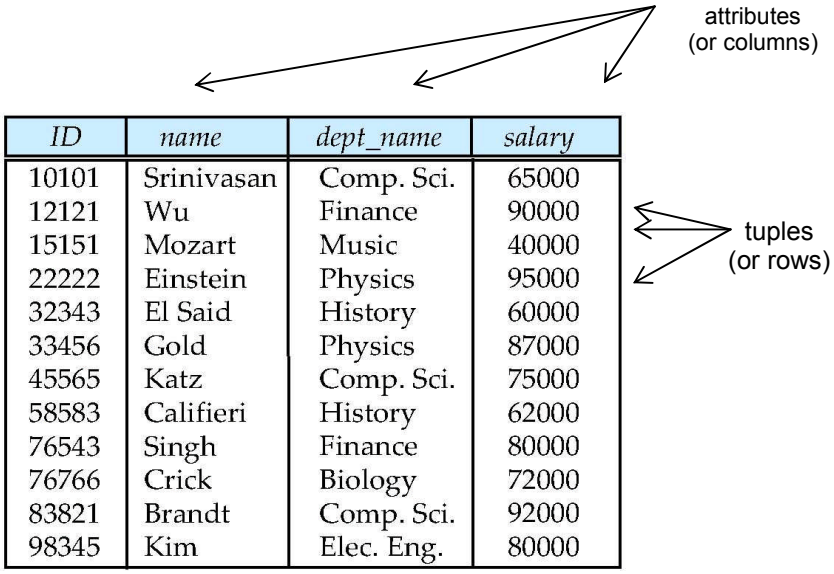
Unit Structure

4.0 INTRODUCTION

In this chapter, we will study the concepts of relation, tuples and attributes. We will further look at the meaning of the term integrity and the various integrity constraints. The relational model is very simple and elegant: a database is a collection of one or more relations, where each relation is a table with rows and columns. This simple tabular representation enables even novice users to understand the contents of a database, and it permits the use of simple, high-level languages to query the data. The major advantages of the relational model over the older data models are its simple data representation and the ease with which even complex queries can be expressed.

4.1 RELATION

A relation is a set of tuples. A database is a collection of relations. A relation is a mathematical entity corresponding to a table. Each row in a table represents a fact that corresponds to and entity or a relationship that exists. Each row is called a tuple. Formally, the column headings of the table are the attributes of a relation.



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 4.1: Relation

4.2 ATTRIBUTE TYPES

The set of allowed values for each attribute is called the domain of the attribute. Attribute values are (normally) required to be atomic; that is, indivisible. The special value null is a member of every domain, indicated that the value is "unknown". The null value causes complications in the definition of many operations. An attribute or a combination of attributes that is used to identify the records uniquely is known as **super key. Candidate key** is defined as minimal super key or irreducible super key; used to identify the records uniquely**.** A candidate key that is used by the database designer for unique identification of each row in a table is known **as primary key.** A primary Key can consist of one or more attributes of a table, known as **composite key**. The candidate key not chosen by database designer as a primary key is known as **alternate key**. **A foreign key** is an attribute or combination of attribute in one table that points to the primary key of another table.

4.3 DOMAIN

A relation is subset of Cartesian product of a list of domains. A table with n attributes must be subset of D1 * D2 * D3 * …… * Dn. A domain can be Atomic or Non-Atomic. Atomic Domains are indivisible. Non-Atomic Domain contains composite values.

4.4 PROPERTIES OF RELATIONS

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

4.5 RELATIONAL MODEL NOTATION

Following notations used in Relational model

1. A relation schema R of degree n is denoted by R(A1,A2,…..An).
2. The uppercase letters Q, R and S denote relation names.
3. The lowercase letters q, r and s denote relation states.
4. The letters t, u and v denote tuples.
5. In general, the name of a relation schema such as EMPLOYEE also indicates the current set of tuples in that relation- the current relation state whereas EMPLOYEES (Eid,Ename,…) refers only to the relation schema.
6. An attribute can be qualified with the relation name R to which it belongs by using the dot notation R.A. For example 'EMPLOYEE.Eid' or 'EMPLOYEE. Ename'. all attribute name in a particular relation must be distinct.

4.6 CHARACTERISTICS OF RELATION

Following are some of the characteristics of relation.

1. Ordering of tuples in a relation:

a) A relation is defined as a set of tuples. Mathematically elements of a set have no order among them hence tuples in a relation do not have any particular order.

b) However, in a file, records are physically stored on disk or in memory, so there always is an order among the records.

c) When we display a relation as a table the rows are displayed in a certain order.

2. Ordering of values within a tuple and an alternative definition of a relation:

a) according to the preceding definition of a relation, an n-tuple is an ordered list of a n values, so the ordering of values in a tuple and hence of attributes in a relation schema is important.

b) however, at a more abstract level, the order of attributes and their values is not that important as long as the correspondence between attributes and values is maintained.

3. Values and NULLs in the tuples:

a) Each value in a tuple is an atomic value that is; it is not divisible into components within the framework of the basic relational model. Hence composite and multivalued attributes are not allowed.

b) This model is sometimes called the flat relational model.

c) Much of the theory behind the relational model was developed with this assumption in mind, which is called the first normal form assumption.

d) Hence multivalued attributes must be represented by separated relations, and composite attributes are represented only by their simple component attributes in the basic relational model.

RELATIONAL CONSTRAINTS

The meaning of constraint is Restriction. There are generally many restrictions or constraints on the actual values in a database state. Constraints on databases can generally be divided into five main categories.

1. Domain constraint
2. Tuple Uniqueness constraint
3. Key constraint
4. Entity Integrity constraint
5. Referential Integrity constraint

1. Domain Constraint

Domain constraint defines the domain or set of values for an attribute. It specifies that the value taken by the attribute must be the atomic value from its domain. The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

2. Tuple Uniqueness constraint

Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.

3. Key constraint

Key constraint specifies that in any relation-All the values of primary key must be unique. The value of primary key must not be null.

4. Entity Integrity constraint

The entity integrity constraint states that primary key value can't be null. This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows. A table can contain a null value other than the primary key field.

5. Referential Integrity constraint

A referential integrity constraint is specified between two tables.

In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

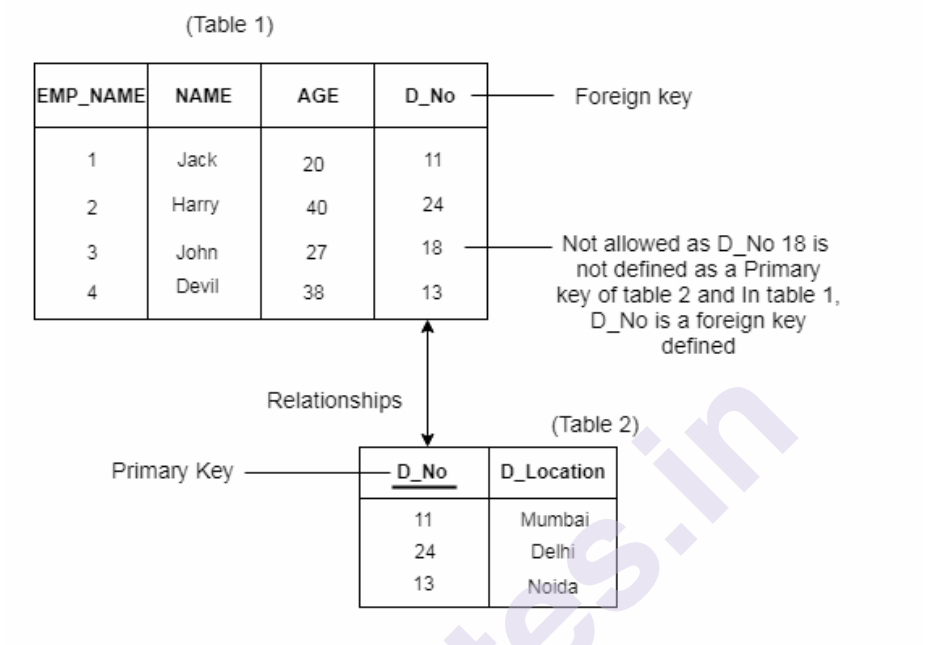


Figure 4.2: Referential integrity

5

ER TO TABLE

Unit Structure
5.1 Rules for converting ER to Table

5.1 RULES FOR CONVERTING ER TO TABLE

1. Convert all the entities in the diagram to tables.
2. All single valued attributes of an entity is converted to a column of the table.
3. Key attribute in the ER diagram becomes the primary key of the table. Declare the foreign key column, if applicable.
4. any multi valued attributes are converted into new table.
5. any composite attributes are merged into same table as different columns. Derived attributes can be ignored.

For example:

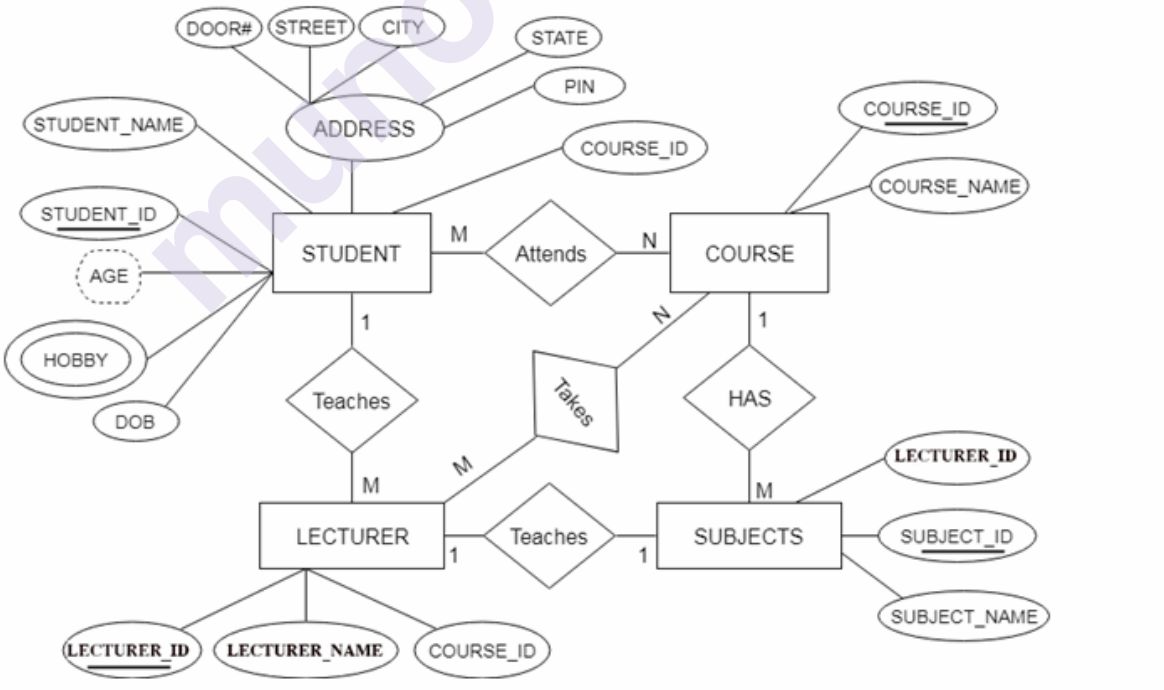


Figure 5.1: ER diagram

There are the following steps which need to be considered before developing table:

Entity type becomes a table:

a) In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.

All single-valued attribute becomes a column for the table:

a) In the STUDENT entity, STUDENT_NAME and STUDENT_ID from the column of STUDENT table.

b) Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.

A key attribute of the entity type represented by the primary key:

a) In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID and LECTURE_ID are the key attribute of the entity.

The multivalued attribute is represented by a separate table:

a) In the student table, a hobby is a multivalued attribute. So, it is not possible to represent multiple values in a single column of STUDENT table.

b) Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.

Composite attribute represented by components:

a) In the given ER diagram student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET and STATE. In the STUDENT table, these attributes can merge as an individual column.

Derived attributes are not considered in the table:

a) In the STUDENT table, age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

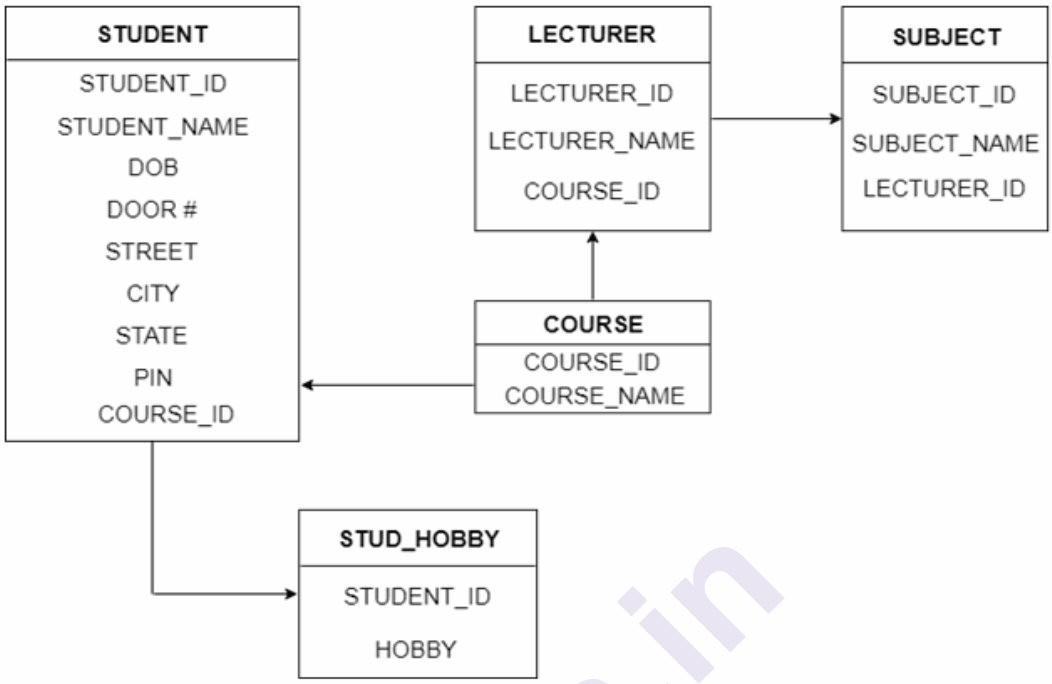


Figure 5.2: table structure of given ER diagram

❖❖❖❖

Unit II

6

SCHEMA REFINEMENT AND NORMAL FORMS

Unit Structure

6.0 OBJECTIVES

In this chapter, we'll look at what functional dependencies are, how to recognise them, and how to infer functional dependencies using inference rules. We'd dig deeper into data normalisation and the various normal forms -1NF,2NF,3NF and BCNF. We'll look at how larger tables can be broken down into smaller ones without losing data at the end of this chapter.

6.1FUNCTIONALDEPENDENCIES

A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes A1, A2, … , An; let us think of the whole database as being described by a single universal. relation schema R = {A1, A2, … , An}. We do not imply that we will actually store the database as a single universal table; we use this concept only in developing the formal theory of data dependencies. A functional dependency is a property of the semantics or meaning of the attributes. The database designers will use their understanding of the semantics of the attributes of R—that is, how they relate to one another—to specify the functional dependencies that should hold on all relation states (extensions) r of R. Relation extensions r(R) that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R.

6.2 NORMALIZATION

The normalization is a process first proposed by Codd in the year 1972. Normalization of data can be considered a process of analysing the

given relation schemas based on their FDs and primary keys to achieve the desirable properties of minimizing redundancy and minimizing the insertion, deletion and update anomalies. It is a process which proceeds in top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary. The goal is to generate a set of relation schemas that allows us to store information without unnecessary redundancy. It also allows easy retrieval of information. The approach is to design schemas that are in an appropriate normal form. To determine whether a relation schema is in one of the desirable normal forms, additional information about the real world is needed to be depicted in the database. Normalisation is based on the functional dependencies.

6.3 TYPES OF NORMAL FORMS

FIRST NORMAL FORM

In the relational model, a domain is atomic if elements of the domain are considered to be indivisible units.

A relation schema R is in first normal form (1NF) if the domains of all attributes of R are atomic.

It is defined to disallow multivalued attributes, composite attributes their combinations, relations within relations or relations as attribute values within tuples. it states that domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. the only attribute values permitted by 1NF are single atomic values. For example, Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

SECOND NORMAL FORM:

A relation schema R is in 2NF, if it satisfies 1NF and if every non-prime attribute A in R is fully functionally dependent on primary key of R. 2NF is based on the concept of full functions dependency. A functional dependency X→Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more. Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF. To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

THIRD NORMAL FORM

If a relation is in 2NF and does not have any transitive partial dependency, it is in 3NF. The 3NF algorithm is used to decrease data duplication. It's also used to ensure data consistency. The relation must be in third normal form if there is no transitive dependency for non-prime characteristics. For every non-trivial function dependency X Y, a relation is in third normal form if it meets at least one of the following conditions.

1. X is a super key
2. Y is a prime property, which means that each of its elements is part of a candidate key.
3.

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

> {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

BCNF (BOYCE CODD NORMAL FORM)

The advanced form of 3NF is BCNF. It's more stringent than 3NF.A table is in BCNF if every functional dependency X → Y, X is the super key of the table.The table should be in 3NF for BCNF, and LHS is super important for every FD.Consider the following scenario: a corporation with workers who work in multiple departments.

EMPLOYEE table:

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

In the above table Functional dependencies are as follows:

```
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
```

Candidate key: {EMP-ID, EMP-DEPT}

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

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

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

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

```
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
```

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

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

6.4 LOSSLESS JOIN DECOMPOSITION

Normalisation leads to decomposition of relation into multiple tables in database. The decomposition should always be lossless to avoid problems like loss of information. Decomposition should guarantee that the join will result in the same relation as it was decomposed. A relational table is decomposed in multiple tables, in such a way that the content of the original table be obtained by joining the decomposed parts. This is called lossless-join or non-additive join decomposition. The lossless join decomposition is defined with respect to functional dependencies.

7

RELATIONAL ALGEBRA

Unit structure:

7.1 Introduction

7.2 Selection

7.3 Projection

7.4 Set operations

7.5 Joins

7.6 Equi join and natural joins

7.1 INTRODUCTION

A procedural query language is relational algebra. It outlines a step-by-step procedure for obtaining the query's result. It performs queries with the help of operators.

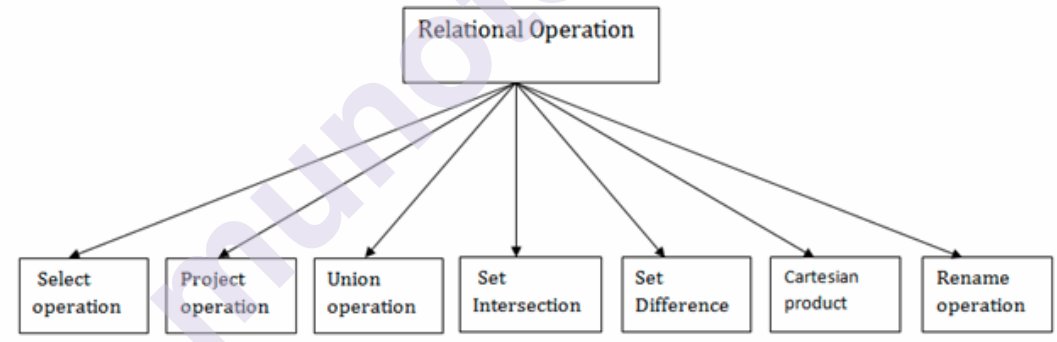


Fig 2.1 : Types Of Operation In Relational Algebra

7.2 SELECTION OPERATION

The select operation finds tuples that match a predicate. It is denoted by sigma (σ).

Notation: σ p(r)

Where:

σ is used for selection prediction

r is used for relation

p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example find all the loan details where branch name is “Perryride”.

σ BRANCH_NAME="perryride" (LOAN)

7.3 PROJECT OPERATION:

This operation displays a list of the properties we want to present in the final product. The remaining attributes are removed from the table. It is denoted by ∏.

Notation: ∏ A1, A2, An (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r**.

For example:

List the names and city of all customers.

∏ NAME, CITY (CUSTOMER)

7.4 SET OPERATION

Union operation:

Assume you have two tuples, R and S. All tuples that are either in R or S, or both in R and S, are included in the union operation**.** It eliminates the duplicate tuples. It is denoted by ∪.

Notation: R ∪ S

The following conditions must be met by a union operation:

- The attribute of the same number must be shared by R and S.
- Duplicate tuples are eliminated automatically.

Consider two relations, BORROW and DEPOSITOR.

∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Set Operation:

Assume you have two tuples, R and S. All tuples in both R and S are included in the set intersection operation.It is denoted by intersection ∩.

Notation: R ∩ S

For example:

Using the above DEPOSITOR table and BORROW table.

∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DE POSITOR)

Set Difference:

Assume you have two tuples, R and S. All tuples that are in R but not in S are included in the set intersection operation. It is denoted by intersection minus (-).

Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table
∏ CUSTOMER_NAME (BORROW) -
∏ CUSTOMER_NAME (DEPOSITOR)

Cartesian product:

Each row in one table is combined with each row in the other table using the Cartesian product. A cross product is another name for it. It is denoted by X.

Notation: E X D

Rename Operation:

The output relation is renamed using the rename method. Rho (ρ) is the symbol for it.For example, we can use the rename operator to rename STUDENT relation to STUDENT1.

ρ(STUDENT1, STUDENT)

7.5 JOINS:

If and only if a specific join condition is satisfied, a Join action joins related tuples from separate relations. It's indicated by⋈.

Operation: (EMPLOYEE ⋈ SALARY)

Types of Joins:

- **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 ⋈.Let's use the EMPLOYEE and SALARY tables.

∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

- **Outer Join:**

The join operation is extended by the outer join operation. It's utilised to deal with information that's lacking.

(EMPLOYEE ⋈ FACT_WORKERS)

An outer join is of three types. Left outer join, right outer join and full outer join.

Left outer join:

The set of tuples in R and S that are equivalent on their shared attribute names is called the left outer join.In the left outer join, tuples in R have no matching tuples in S. It is denoted by ⍰.Using the above EMPLOYEE table and FACT_WORKERS table.

EMPLOYEE ⍰ FACT_WORKERS

Right outer join:

The set of tuples in R and S that are equivalent on their shared attribute names is called the right outer join.The set of tuples in R and S that are equivalent on their shared attribute names is called the right outer join.It is denoted by ⍰.Using the above EMPLOYEE table and FACT_WORKERS Relation.

EMPLOYEE ⍰ FACT_WORKERS

Full outer join:

The full outer join is like a left or right join, except it includes all rows from both tables.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 are used in full outer join.It is denoted by ⍰.Using the above EMPLOYEE table and FACT_WORKERS table.

EMPLOYEE ⍰ FACT_WORKERS

7.6 EQUI-JOIN:

An inner join is another name for it. It's the most prevalent type of connection. It is based on data that has been matched according to the equality criteria. The comparison operator is used in the equi join.

CUSTOMER ⋈ PRODUCT

❖❖❖❖

8

DDL STATEMENTS

Unit structure:

8.1 Creating Databases

8.2 Using Databases

8.3 Creating Tables with integrity constraints

8.4 Altering Tables

8.5 Renaming Tables

8.6 Dropping Tables

8.7 Backing Up and Restoring databases

8.1 CREATING DATABASES:

SQL DDL commands are used to create schemas and tables and gives an overview of basic data types used in creating a database. SQL uses some terms such as table, row and column which are knows as relation, tuple and attribute respectively. The basic command is CREATE command. It can not only create table but also schemas, domains and views.

CREATE DATABASE Database_Name;

In this syntax, **Database_Name** specifies the name of the database which we want to create in the system. Just after the 'Create Database' keyword, we must type the database name in the query. The database we wish to make should have a clear and distinct name that can be easily recognised. The name of the database should be no more than 128 characters long.

CREATE TABLE command: In a database, the Construct TABLE statement is used to create tables.If you wish to make a table, you'll need to give it a name and specify each column's data type.

Create table "tablename"
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
For example:
Create table Employee(
Eid varchar2(20),
Enamechar(30))

DATATYPES:

CHAR(Size)	It is used to specify a fixed length string that can contain numbers, letters, and special characters. Its size can be 0 to 255 characters. Default is 1.
VARCHAR(Size)	It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.
BINARY(Size)	It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1.
VARBINARY(Size)	It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the maximum column length in bytes.
TEXT(Size)	It holds a string that can contain a maximum length of 255 characters.
TINYTEXT	It holds a string with a maximum length of 255 characters.
MEDIUMTEXT	It holds a string with a maximum length of 16,777,215.
LONGTEXT	It holds a string with a maximum length of 4,294,967,295 characters.
ENUM(val1, val2, val3,...)	It is used when a string object having only one value, chosen from a list of possible values. It contains 65535 values in an ENUM list. If you insert a value that is not in the list, a blank value will be inserted.
SET(val1,val2,val3,....)	It is used to specify a string that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values at one time in a SET list.
BLOB(size)	It is used for BLOBs (Binary Large Objects). It can hold up to 65,535 bytes.

8.3 CREATING TABLES WITH INTEGRITY CONSTRAINTS:

- **NOT NULL** − Ensures that a column cannot have NULL value.
- **DEFAULT** − Provides a default value for a column when none is specified.
- **UNIQUE** − Ensures that all values in a column are different.
- **PRIMARY Key** − Uniquely identifies each row/record in a database table.
- **FOREIGN Key** − Uniquely identifies a row/record in any of the given database table.
- **CHECK** − The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- **INDEX** − Used to create and retrieve data from the database very quickly.

8.4 ALTERING TABLES:

ALTER TABLE command is used to add, delete or modify columns in an existing table. You should also use the ALTER TABLE command to add and drop various constraints on an existing table. The basic syntax of an ALTER TABLE command to add a New Column in an existing table is as follows.

ALTER TABLE table_name ADD column_namedatatype;

The basic syntax of an ALTER TABLE command to DROP COLUMN in an existing table is as follows.

ALTER TABLE table_name DROP COLUMN column_name;

DROP TABLE statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

The basic syntax of this DROP TABLE statement is as follows −

DROP TABLE table_name;

8.5 RENAME OPERATION:

ALTER TABLE table_name RENAME TO new_table_name;

8.6 BACKING UP AND RESTORING DATA:

Reasons of Failure in a Database

There can be multiple reasons of failure in a database because of which a database backup and recovery plan is required. Some of these reasons are:

- **User Error -** Normally, user error is the biggest reason of data destruction or corruption in a database. To rectify the error, the database needs to be restored to the point in time before the error occurred.
- **Hardware Failure -** This can also lead to loss of data in a database. The database is stored on multiple hard drives across various locations. These hard drives may sometimes malfunction leading to database corruption. So, it is important to periodically change them.
- **Catastrophic Event -** A catastrophic event can be a natural calamity like a flood or earthquake or deliberate sabotage such as hacking of the database. Either way, the database data may be corrupted, and backup may be required.

Methods of Backup

The different methods of backup in a database are:

- **Full Backup -** This method takes a lot of time as the full copy of the database is made including the data and the transaction records.
- **Transaction Log -** Only the transaction logs are saved as the backup in this method. To keep the backup file as small as possible, the previous transaction log details are deleted once a new backup record is made.
- **Differential Backup -** This is similar to full back up in that it stores both the data and the transaction records. However only that information is saved in the backup that has changed since the last full backup. Because of this, differential backup leads to smaller files.

Database Recovery

There are two methods that are primarily used for database recovery. These are:

- **Log based recovery -** In log-based recovery, logs of all database transactions are stored in a secure area so that in case of a system failure, the database can recover the data. All log information, such as the time of the transaction, its data etc. should be stored before the transaction is executed.
- **Shadow paging -** In shadow paging, after the transaction is completed, its data is automatically stored for safekeeping. So, if the system crashes in the middle of a transaction, changes made by it will not be reflected in the database.

9

DML STATEMENTS

Unit Structure

9.1 DML COMMANDS

DML commands are used to modify the database. It is responsible for all form of changes in the database.The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE_NAME
(col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
```

For example:

```
INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");
```

UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name SET [column_name1= value1,...column_na
meN = valueN] [WHERE CONDITION]
```

For example:

```
UPDATE students
SET User_Name = 'Sonoo'
WHERE Student_Id = '3'
```

DELETE: It is used to remove one or more row from a table.

Syntax:

```
DELETE FROM table_name [WHERE condition];
```

For example:

```
DELETE FROM javatpoint
WHERE Author="Sonoo";
```

9.2 CONDITIONAL SELECT

SELECT statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

The basic syntax of the SELECT statement is as follows −

SELECT column1, column2, columnN FROM table_name;

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

SELECT * FROM table_name;

For example:

Select * from employee;

WHERE clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records. The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc.

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

SELECT column1, column2, columnN

FROM table_name

WHERE [condition]

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc.

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 −

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY >2000;
```

This would produce the following result −

ID	**NAME**	**SALARY**
4	Chaitali	6500.00
5	Hardik	8500.00
6	Komal	4500.00
7	Muffy	10000.00

9.3 IN OPERATOR (SET MEMBERSHIP TEST):

The IN conditional operator actually performs a set membership test. To put it another way, it's used to see if a value (expressed before the keyword IN) is "in" the list of values provided after the keyword IN. For example

```
SELECT employeeid, lastname, salary
FROM employee_info
WHERE lastname IN ('Hernandez', 'Jones', 'Roberts', 'Ruiz');
```

This statement will select the employeeid, lastname, salary from the employee_info table where the lastname is equal to either: Hernandez, Jones, Roberts, or Ruiz. It will return the rows if it is ANY of these values.

9.4 BETWEEN (RANGE TEST):

The BETWEEN conditional operator is used to test to see whether or not a value (stated before the keyword BETWEEN) is "between" the two values stated after the keyword BETWEEN.For example:

```
SELECT employeeid, age, lastname, salary
FROM employee_info
WHERE age BETWEEN 30 AND 40;
```

This statement will select the employeeid, age, lastname, and salary from the employee_info table where the age is between 30 and 40 (including 30 and 40).

9.5 ORDER BY clause:

ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

The basic syntax of the ORDER BY clause is as follows −

SELECT column-list

FROM table_name

[WHERE condition]

[ORDER BY column1, column2, ..columnN] [ASC | DESC];

In the ORDER BY clause, you can utilise more than one column. Make sure that whichever column you're using to sort is included in the column-list. For example:

SQL> SELECT * FROM CUSTOMERS

ORDER BY NAME, SALARY;

9.6 GROUP BY operator:

GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

SELECT column1, column2

FROM table_name

WHERE [conditions]

GROUP BY column1, column2

ORDER BY column1, column2

Consider the CUSTOMERS table is having the following records −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

SQL>SELECT NAME, SUM(SALARY) FROM CUSTOMERS
GROUP BY NAME;

This would produce the following result −

NAME	SUM(SALARY)
Chaitali	6500.00
Hardik	8500.00
kaushik	2000.00
Khilan	1500.00
Komal	4500.00
Muffy	10000.00
Ramesh	2000.00

9.7 AGGREGATE FUNCTION:

Aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.It is also used to summarize the data. There are five aggregate functions which are follows:

1) COUNT
2) MAX
3) MIN
4) AVG
5) SUM

1. COUNT FUNCTION

COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax: COUNT(*) or COUNT([ALL|DISTINCT] expression)

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax: SUM() or SUM([ALL|DISTINCT] expression)

Example: SUM()

SELECT SUM(COST) FROM PRODUCT_MAST;

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax: AVG() or AVG([ALL|DISTINCT] expression)

Example:

```
SELECT AVG(COST)
FROM PRODUCT_MAST;
```

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column. Syntax:
MAX() or MAX([ALL|DISTINCT] expression)

Example:

```
1. SELECT MAX(RATE)
2. FROM PRODUCT_MAST;
```

5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.
SyntaxMIN() or MIN([ALL|DISTINCT] expression)
Example:

```
SELECT MIN(RATE)
FROM PRODUCT_MAST;
```


Unit - III

10

FUNCTINS, JOIN, SUBQUERIES, VIEWS, DATA PROTECTION AND DCL STATEMENTS

Unit Structure

10.0 OBJECTIVES: -

After going through this unit, you will be able to:

- Learn functions in SQL, like math string and date
- state the DCL statements in SQL
- describe the basic concepts in views, subqueries, join and system privilege
- illustrate the role of a DBA

10.1 INTRODUCTION: -

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Views can join and simplify multiple tables into a single virtual table. Views can act as aggregated tables, where the database engine aggregates data (sum, average, etc.) and presents the calculated results as part of the data. Views can hide the complexity of data.

SQL functions are sub-programs, which are commonly used and re-used throughout SQL database applications for processing or manipulating data. All SQL database systems have DDL (data definition language) and DML (data manipulation language) tools to support the creation and maintenance of databases.

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

A privilege is a right to execute a particular type of SQL statement or to access another user's object. Some examples of privileges include the right to: Connect to the database (create a session) Create a table.

10.2 FUNCTIONS: -

10.2.1 String Functions:

CONCAT:
It merges two or more strings or a string and a data value together
Example:
SELECT CONCAT('summer ','18') FROM DUAL;

INSTR:
The INSTR() function returns the position of the first occurrence of a string in another string.
Example:
SELECT INSTR("RamSham.com", "3") AS MatchPosition;

LEFT:
This function returns the leftmost n characters from the string str. If the string is empty, it returns NULL.
Example:
SELECT LEFT('RamSham', 4);

RIGHT:
This function returns the rightmost n characters from the string str. If the string is empty, it returns NULL.
Example:
SELECT RIGHT('RamSham', 5);

MID:
The MID() function extracts a substring from a string (starting at any position).
Example:
SELECT MID("SQL Tutorial", 5, 3) AS ExtractString;

LENGTH:
Find outs the length of given string.
Example:
SELECT LENGTH (‘abcd’) FROM DUAL

LOWER:
Converts a string to all lowercase characters.
Example:
SELECT LOWER (‘ABCD’) FROM DUAL

UPPER:
Converts a string to all uppercase characters.
Example:
SELECT UPPER (‘abcd’) FROM DUAL

REPLACE:
It returns character string with each occurrence of search string replaced with [repstring]
Example:
SELECT REPLACE (‘Tick and Tock’, ‘T’,’C’) FROM DUAL

STRCMP:
This function compares both the strings str1 and str2. It returns 0 if both strings are equal, 1 if str1 is greater than str2 and -1 if if str2 is greater than str1.

Example:
SELECT STRCMP('HARRY', 'HARRY');

TRIM:
The TRIM() function removes the space character OR other specified characters from the start or end of a string.
Example:
SELECT TRIM('#! ' FROM ' #SQL Tutorial! ') AS TrimmedString;

LTRIM:
Removesleading spaces from a string
Example:
SELECT LTRIM (‘abcd) FROM DUAL;

RTRIM:
Removes trailing spaces from a string
Example:
SELECT RTRIM (‘abcd’) FROM DUAL;

10.2.2 Math Functions

ABS:
This function returns the absolute value of X.
Example:
Select abs(-6);

CEIL:
This returns the smallest integer value that is either more than X or equal to it.
Example:
SELECT CEIL(5.7);

FLOOR:
This returns the largest integer value that is either less than X or equal to it.
Example:
SELECT FLOOR(5.7);

MOD:
The variable X is divided by Y and their remainder is returned.
Example:
SELECT MOD(9,5);

POW:
This function returns the value of x raised to the power of Y
Example:
SELECT POWER(2,5);

SQRT:
This function returns the square root of X.
Example:
SELECT SQRT(9);

ROUND:
This function returns the value of X rounded off to the whole integer that is nearest to it.
Example:
SELECT ROUND(5.7);

10.2.3 Date Functions :-

ADDDATE:
ADDDATE() is a synonym for DATE_ADD().
Example:
SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);

DATEDIFF:
DATEDIFF() returns expr1 . expr2 expressed as a value in days from one date to the other. Both expr1 and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.
Example :-
SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');

DAY:
The DAY() is a synonym for the DAYOFMONTH() function. Returns the day of the month for date, in the range 0 to 31.
Example:
SELECT DAYOFMONTH('1998-02-03');

MONTH:
Returns the month for date, in the range 0 to 12.
Example:
SELECT MONTH('1998-02-03')

YEAR:
Returns the year for date, in the range 1000 to 9999, or 0 for the .zero. date.
Example:
SELECT YEAR('98-02-03');

HOUR:
Returns the hour for time. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.
Example:
SELECT HOUR('10:05:03');

MIN:
Returns the minute for time, in the range 0 to 59.
Example:
SELECT MINUTE('98-02-03 10:05:03');

SEC:
Returns the second for time, in the range 0 to 59.
Example:
SELECT SECOND('10:05:03');

NOW:
Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. This value is expressed in the current time zone.
Example:
SELECT NOW();

REVERSE:
The REVERSE() function reverses a string and returns the result.
Example:
SELECT REVERSE('SQL Tutorial');

10.3 JOINS

- Joins are used to relate information in different tables.
- A Join condition is a part of the sql query that retrieves rows from two or more tables.
- A SQL Join condition is used in the SQL WHERE Clause of select, update, delete statements.

Syntax for joining two tables is:
SELECT col1, col2, col3...
FROM table_name1, table_name2
WHERE table_name1.col2 = table_name2.col1;

If a sql join condition is omitted or if it is invalid the join operation will result in a Cartesian product. The Cartesian product returns a number of rows equal to the product of all rows in all the tables being joined.

Example:
If the first table has 20 rows and the second table has 10 rows, the result will be 20 * 10, or 200 rows.
This query takes a long time to execute.

Let us use the below two tables to explain the sql join conditions.

Database table "product";

Product_id	Product_name	Supplier_name	Unit_price
100	Camera	Nikon	300
101	Television	LG	100
102	Refrigerator	Videocon	150
103	IPod	Apple	75
104	Mobile	Nokia	50

Database table "order_items";

order_id	product_id	total_units	customer
5100	104	30	Infosys
5101	102	5	Satyam
5102	103	25	Wipro
5103	101	10	TCS

Joins can be classified into Equi join and Non Equi join.

1. SQL Equi joins
2. SQL Non equi joins

10.3.1 SQLEqui joins

It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of Equijoins are SQL Outer join and SQL Inner join.

Example:
We can get Information about a customer who purchased a product and the quantity of product.

An Equi-join is classified into two categories:

a) SQL Inner Join
b) SQL Outer Join

a) SQL Inner Join:

All the rows returned by the sql query satisfy the sql join condition specified.

Example:
To display the product information for each order the query will be as given below.

Since retrieving the data from two tables, you need to identify the common column between these two tables, which is the product_id.

QUERY:
SELECT order_id, product_name, unit_price, supplier_name, total_units
FROM product, order_items
WHERE order_items.product_id = product.product_id;

The columns must be referenced by the table name in the join condition, because product_id is a column in both the tables and needs a way to be identified.

b) SQL Outer Join:

- Outer join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables.
- The syntax differs for different RDBMS implementation.
- Few of them represent the join conditions as" LEFT OUTER JOIN" and "RIGHT OUTER JOIN".

Example
Display all the product data along with order items data, with null values displayed for order items if a product has no order item.

QUERY
SELECT p.product_id, p.product_name, o.order_id, o.total_units
FROM order_items o, product p
WHERE o.product_id (+) = p.product_id;

Output:

Product_id	product_name	order_id	total_units
100	Camera		
101	Television	5103	10
102	Refrigerator	5101	5
103	IPod	5102	25

SQL Self Join:
A Self Join is a type of sql join which is used to join a table to it, particularly when the table has a FOREIGN KEY that references its own PRIMARY KEY.

It is necessary to ensure that the join statement defines an alias for both copies of the table to avoid column ambiguity.

Example
SELECT a.sales_person_id, a.name, a.manager_id, b.sales_person_id, b.name
FROM sales_person a, sales_person b
WHERE a.manager_id = b.sales_person_id;

10.3.2 SQL NON-EQUI JOIN:

A Non Equi Join is a SQL Join whose condition is established using all comparison operators except the equal (=) operator.
Like >=, <=, <, >

Example:
Find the names of students who are not studying either Economics, the sql query would be like, (let's use Employee table defined earlier.)

QUERY:
SELECT first_name, last_name, subject FROM Employee
WHERE subject != 'Economics'

Output:

first_name	last_name	subject
Anajali	Bhagwat	Maths
Shekar	Gowda	Maths
Rahul	Sharma	Science
Stephen	Fleming	Science

10.4 SUBQUERIES

A subquery is a SELECT statement with another SQL statement, like in the example below.

SELECT *
FROM product
WHERE id IN (SELECT product_idFROM provider_offer WHERE provider_id = 156);

Subqueries are further classified as either a correlated subquery or a nested subquery. They are usually constructed in such a way to return:

a table

SELECT MAX(average.average_price)
FROM (SELECTproduct_category,
AVG(price) AS average_price FROM product GROUP BY product_category) average;

or a value

SELECT id
FROM purchase
WHERE value >(SELECT AVG(value) FROM purchasec);

10.4.1 NestedSubqueries: -
Nested subqueries are subqueries that don't rely on an outer query. In other words, both queries in a nested subquery may be run as separate queries.

This type of subquery could be used almost everywhere, but it usually takes one of these formats:

SELECT
FROM
WHERE [NOT] IN (subquery)
SELECT *FROM clientWHERE city IN (SELECT city FROM provider);

The example subquery returns all clients that are FROM the same city as the product providers.

The **IN** operator checks if the value is within the table and retrieves the matching rows.

SELECTFROMWHERE expression comparison_operator [ANY| ALL] (subquery)

10.4.2Subquery With ALL Operator:
The ALL operator compares a value to every value FROM the result table.

For example, the following query returns all of the models and producers of bikes that have a price greater than the most expensive headphones.

SELECT producer, model
FROM product
WHERE product_category = 'bike'
AND price >ALL(SELECT price FROM product WHERE product_category = 'headphones');

Similar subquery but with ANY operator:

10.4.3 Subquery With ANY Operator:
The ANY operator compares a value to each value in a table and evaluates whether or not the result of an inner query contains at least one row.

The following query returns all of the models and producers of bikes that have a price greater than at least one of the headphones.

SELECT producer, model
FROM product
WHERE product_category = 'bike'

AND price >ANY(SELECT price FROM product WHERE product_category = 'headphones');

You can also nest a subquery in another subquery. For example:

Subquery Nested in Another Subquery Using IN Operator:
This query returns producers and models of bikes that exist in provider's offers FROM the USA.

```
SELECT producer, model
FROM product
WHERE product_category = 'bike'
AND id IN (SELECT distinct product_idFROMprovider_offer
WHERE provider_id IN (SELECT id FROM provider WHERE country = 'USA' ) );
```

The same could be done using joins.

```
SELECT product.producer, product.model
FROM product, provider_offer, provider
WHERE provider_offer.product_id = product.id
  AND provider_offer.provider_id = provider.id
  AND product_category = 'bike'
  AND provider.country = 'USA';
```

10.4.4 CorrelatedSubqueries:-
Subqueries are correlated when the inner and outer queries are interdependent, that is, when the outer query is a query that contains a subquery and the subquery itself is an inner query. Users that know programming concepts may compare it to a nested loop structure.

Let's start with a simple example.

The inner query calculates the average value and returns it. In the outer query's WHERE clause, we filter only those purchases which have a value greater than the inner query's returned value.

Subquery Correlated in **WHERE** Clause

```
SELECT id
FROM purchase p1
WHERE date > '2013-07-15'
  AND value >( SELECT AVG(value) FROM purchase p2 WHERE p1.date = p2.date  );
```

The query returns purchases after 15/07/2014 with a total price greater than the average value FROM the same day.

The equivalent example, but with joining tables.

```
SELECT  p1.id
FROM purchase p1, purchase p2
WHERE p1.date = p2.date
  AND p1.date> '2013-07-15'
GROUP BY p1.idHAVING p1.value > AVG(p2.value);
```

This example can also be written as a SELECT statement with a subquery correlated in a FROM clause.

The subquery returns the table that contains the average value for each purchase for each day. We join this result with the Purchase table on column ‘date’ to check the condition date > ’15/07/2014'.

```
SELECT id
FROM
  purchase,
  (
    SELECT date, AVG(value) AS average_value
    FROM purchase
    WHERE date > '2013-07-15'
    GROUP BY date
  ) average
WHERE purchase.date  = average.date
  AND purchase.date> '2013-07-15'
  AND purchase.value>average.average_value;
```

Usually, this kind of subquery should be avoided because indexes can’t be used on a temporary table in memory.

10.4.5 Subquery With EXISTS:-

The EXISTS operator checks if the row FROM the subquery matches any row in the outer query. If there’s no data matched, the EXISTS operator returns FALSE.

Syntax

```
SELECTFROMWHERE [NOT] EXISTS (subquery)
```

Example:

This Query returns all clients that ordered after 10/07/2013.

```
SELECT id, company_name
FROM client
WHERE EXISTS(
  SELECT *
  FROM purchase
  WHERE client.id = purchase.client_id
  WHERE date > '2013-07-10'
);
```

When a subquery is used, the query optimizer performs additional steps

before the results FROM the subquery are used. If a query that contains a subquery can be written using a join, it should be done this way. Joins usually allow the query optimizer to retrieve the data in a more efficient way.

10.4.6 Subquery Restrictions:-

A subquery is subject to these restrictions:

- The subquery_select_list can consist of only one column name, except in the exists subquery, where an (*) is usually used in place of the single column name. You can use an asterisk (*) in a nested select statement that is not an exists subquery.
- Do not specify more than one column name. Qualify column names with table or view names if there is ambiguity about the table or view to which they belong.
- Subqueries can be nested inside the WHERE or HAVING clause of an outer select, insert, update, or delete statement, inside another subquery, or in a select list. Alternatively, you can write many statements that contain subqueries as joins; Adaptive Server processes such statements as joins.
- In Transact-SQL, a subquery can appear almost anywhere an expression can be used, if it returns a single value. SQL derived tables can be used in the from clause of a subquery wherever the subquery is used.
- You cannot use subqueries in an order by, group by, or compute by list.
- You cannot include a for browse clause in a subquery.
- You cannot include a union clause in a subquery unless it is part of a derived table expression within the subquery.
- The select list of an inner subquery introduced with a comparison operator can include only one expression or column name, and the subquery must return a single value. The column you name in the where clause of the outer statement must be join-compatible with the column you name in the subquery select list.
- You cannot include text, unitext, or image datatypes in subqueries.
- Subqueries cannot manipulate their results internally, that is, a subquery cannot include the order by clause, the compute clause, or the into keyword.
- Correlated (repeating) subqueries are not allowed in the select clause of an updatable cursor defined by declare cursor.
- There is a limit of 50 nesting levels.
- The maximum number of subqueries on each side of a union is 50.

- The where clause of a subquery can contain an aggregate function only if the subquery is in a having clause of an outer query and the aggregate value is a column from a table in the from clause of the outer query.
- The result expression from a subquery is subject to the same limits as for any expression. The maximum length of an expression is 16K.

10.5 DATABASE PROTECTION:

Database security is the protection of the database against intentional and unintentional threats that may be computer-based or non-computer-based. Database security is the business of the entire organization as all people use the data held in the organization's database and any loss or corruption to data would affect the day-to-day operation of the organization and the performance of the people. Therefore, database security encompasses hardware, software, infrastructure, people and data of the organization.

Now there is greater emphasis on database security than in the past as the amount of data stored in corporate database is increasing and people are depending more on the corporate data for decision-making, customer service management, supply chain management and so on. Any loss or unavailability to the corporate data will cripple today's organization and will seriously affect its performance. Now the unavailability of the database for even a few minutes could result in serious losses to the organization.

10.5.1 Security Issues

Database security is a broad area that addresses many issues, including the following:

- **Various legal and ethical issues regarding the right to access certain information.**

  for example, some information may be deemed to be private and can-not be accessed legally by unauthorized organizations or persons. In the United States, there are numerous laws governing privacy of information.

- **Policy issues at the governmental, institutional, or corporate level as to what kinds of information should not be made publicly available.**

  for example, credit ratings and personal medical records.

- **System-related issues such as the system levels at which various security functions should be enforced.**

  for example, whether a security function should be handled at the

physical hardware level, the operating system level, or the DBMS level.

- **The need in some organizations to identify multiple security levels and to categorize the data and users based on these classifications.**

  for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications of data must be enforced.

10.5.2 Threats to Databases.

Threats to databases can result in the loss or degradation of some or all of the following commonly accepted security goals: integrity, avail-ability, and confidentiality.

- **Loss of integrity.** Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, updating, changing the status of data, and deletion. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.
- **Loss of availability.** Database availability refers to making objects available to a human user or a program to which they have a legitimate right.
- **Loss of confidentiality.** Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security. Unauthorized, unanticipated, or unintentional disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.

To protect databases against these types of threats, it is common to implement four kinds of control measures: access control, inference control, flow control, and encryption.

10.5.3 Security Mechanisms

To protect the database, we must take security measures at several levels:

- **Physical:** The sites containing the computer systems must be secured against armed or surreptitious entry by intruders.
- **Human:** Users must be authorized carefully to reduce the chance of any such user giving access to an intruder in exchange for a bribe or other favours.
- **Operating System:** No matter how secure the database system is,

weakness in operating system security may serve as a means of unauthorized access to the database.

- **Network:** Since almost all database systems allow remote access through terminals or networks, software-level security within the network software is as important as physical security, both on the Internet and in networks private to an enterprise.
- **Database System:** Some database-system users may be authorized to access only a limited portion of the database. Other users may be allowed to issue queries, but may be forbidden to modify the data. It is responsibility of the database system to ensure that these authorization restrictions are not violated.

10.5.4 Discretionary security mechanisms.
These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).

Discretionary Access Control Based on Granting and Revoking Privileges
The typical method of enforcing discretionary access control in a database system is based on the granting and revoking of privileges. Let us consider privileges in the context of a relational DBMS. In particular, we will discuss a system of privileges somewhat similar to the one originally developed for the SQL language (see Chapters 4 and 5). Many current relational DBMSs use some variation of this tech-nique. The main idea is to include statements in the query language that allow the DBA and selected users to grant and revoke privileges.

1. Types of Discretionary Privileges
In SQL2 and later versions, the concept of an authorization identifier is used to refer, roughly speaking, to a user account (or group of user accounts). For simplicity, we will use the words user or account interchangeably in place of authorization identifier. The DBMS must provide selective access to each relation in the database based on specific accounts. Operations may also be controlled; thus, having an account does not necessarily entitle the account holder to all the functionality provided by the DBMS. Informally, there are two levels for assigning privileges to use the database system:

- **The account level.** At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- **The relation (or table) level.** At this level, the DBA can control the privilege to access each individual relation or view in the database.
- **References privilege on R.** This gives the account the capability to reference (or refer to) a relation R when specifying integrity

constraints. This privilege can also be restricted to specific attributes of R.

Notice that to create a view, the account must have the SELECT privilege on all relations involved in the view definition in order to specify the query that corresponds to the view.

2. Specifying Privileges through the Use of Views

The mechanism of views is an important discretionary authorization mechanism in its own right. For example, if the owner A of a relation R wants another account B to be able to retrieve only some fields of R, then A can create a view V of R that includes only those attributes and then grant SELECT on V to B. The same applies to limiting B to retrieving only certain tuples of R; a view V can be created by defining the view by means of a query that selects only those tuples from R that A wants to allow B to access.

3. Revoking of Privileges

In some cases it is desirable to grant a privilege to a user temporarily. For example, the owner of a relation may want to grant the SELECT privilege to a user for a specific task and then revoke that privilege once the task is completed. Hence, a mechanism for revoking privileges is needed. In SQL a REVOKE command is included for the purpose of cancelling privileges.

4. Propagation of Privileges Using the GRANT OPTION

Whenever the owner A of a relation R grants a privilege on R to another account B, the privilege can be given to B with or without the GRANT OPTION. If the GRANT OPTION is given, this means that B can also grant that privilege on R to other accounts. Suppose that B is given the GRANT OPTION by A and that B then grants the privilege on R to a third account C, also with the GRANT OPTION. In this way, privileges on R can propagate to other accounts without the knowledge of the owner of R. If the owner account A now revokes the privilege granted to B, all the privileges that B propagated based on that privilege should automatically be revoked by the system.

It is possible for a user to receive a certain privilege from two or more sources. For example, A4 may receive a certain UPDATE R privilege from both A2 and A3. In such a case, if A2 revokes this privilege from A4, A4 will still continue to have the privilege by virtue of having been granted it from A3. If A3 later revokes the privilege from A4, A4 totally loses the privilege. Hence, a DBMS that allows propagation of privi-leges must keep track of how all the privileges were granted so that revoking of priv-ileges can be done correctly and completely.

Role of DBA

A person having who has central control over data and programs that access the data is called DBA. Following are the functions of the DBA.

- **Schema definition:** DBA creates database schema by executing Data Definition Language (DDL) statements.
- **Storage** structure and **access** method definition
- **Schema and physical organization modification:** If any changes are to be made in the original schema, to fit the need of your organization, then these changes are carried out by the DBA.
- **Granting of authorization for data access:** DBA can decide which parts of data can be accessed by which users. Before any user access the data, DBMS checks which rights are granted to the user by the DBA.
- **Routine maintenance:** DBA has to take periodic backups of the database, ensure that enough disk space is available to store new data, ensure that performance of DBMS ix not degraded by any operation carried out by the users.
- **Performance monitoring:** Here DBMS should respond to changes in requirements, i.e.,changing details of storage and access thereby organising the system so as to get the performance that is ` best for the enterprise'.

10.6 VIEWS: -

Definition:

- A view is a virtual table that consists of columns from one or more tables.
- A virtual table is like a table containing fields but it does not contain any data. In run time it contains the data and after that it gets free.
- But table stores the data in database occupy some space.
- Just like table, view contains Rows and Columns which is fully virtual based table.
- **Base Table** -The table on which view is defined is called as Base table.

10.6.1 Creating a VIEW

This statement is used to create a view.

Syntax:

CREATE VIEW view_name

- The CREATE statement assigns a name to the view and also gives the query which defines the view.
- To create the view, one should must have privileges to access all of the base tables on which view is defined.

- The create view can change name of column in view as per requirements.

Horizontal View

A Horizontal view will restrict the user's access to only a few rows of the table.

Example:

Define a view for Sue (employee number 1004) containing only orders placed by customers assigned to her.

CREATE VIEW SUEORDERS AS SELECT *

FROM ORDERS WHERE CUST IN

(SELECT CUST_NUM FROM CUSTOMERS WHERE CUST_REP=1004)

Vertical View

A vertical view restricts a user's access to only certain columns of a table.

Ex:

CREATE VIEW EMP_ADDRESS AS

SELECT EMPNO, NAME, ADDR1, ADDR2, CITY FROM EMPLOYEE

ROW/COLUMN SUBSET VIEW.

- Views can be used to restrict a user to access only selected set of rows and columns of a table in a database.
- This view generally helps us to visualize how view can represent the base table.
- This type of view is combination of both horizontal and vertical views.

Ex:

CREATE VIEW STUDENTS_PASSED AS SELECT ROLLNO, NAME, PERCENTAGE

FROM STUDENTS

WHERE RESULT ='PASS'

Grouped View

- A grouped view is one in which query includes GROUPBY CLAUSE.
- It is used to group related rows of data and produce only one result row for each group.

Ex:

Find summary information of Employee Salaries in sales Department.

CREATE VIEW Summary_Empl_Sal (

Total_Employees, Minimum_salary, Maximum_Salary, Average_salary, Total_salary)

AS

SELECT COUNT(EmpID),

Min(Salary), Max(Salary), Avg(Salary), SUM(Salary), FROM Employee

GROUP BY Department HAVING Department='Sales';

View Call

SELELCT *FROM Summary_Empl_Sal

The above Query will give,

Total No. Of Employees in sales Department, Minimum Salary in sales Department.

Maximum Salary in sales Department. Average Salary in sales Department.

Total Salary of Employees in sales Department.

Joined Views

- A Query based on more than one base table is called as Joined View.
- It is also called as Complex View
- This gives a way to simplify multi table queries by joining two or more table query in the view definition that draws its data from multiple tables and presents the query results as a single view.
- The view once it is ready we can retrieve data from multiple tables without joining any table simply by accessing a view created.

Ex:

Company database find out all EMPLOYEES for respective DEPARTMENTS.

CREATE VIEW Emp_Details As

Select Employee,EmpID, Department, DeptID, Department, DeptName From

Where Employee.DeptID=Department.DeptID;

View Call

SELECT * FROMEmp_Details

10.6.2 DROPPING VIEW

When a view is no longer needed, it can be removed by using DROP VIEW statement.

Syntax:

DROP VIEW <VIEW NAME> [CASCADE/RESTRICT]

CASCADE: It deletes the view with all dependent view on original view.

RESTRICT: It deletes the view only if they're in no other view depends on this view.

Example:

Consider that we have view VABC and VPQR. ViewVPQR depends on VABC.

Query:

DROP view VABC

If we drop VABC, then cascading affect takes place and view VPQR is also dropped.

Thus, default option for dropping a view is CASCADE. The CASCADE option tells DBMS to delete not only the named view, but also query views that depend on its definition. But,

QUERY:

DROP view VABC RESTRICT

Here, the query will fail because of RESTRICT option tells DBMS to remove the view only if no other views depend on it. Since VPQR depends on VABC, will cause an error.

10.6.3 UPDATING VIEWS

- Records can be updated, inserted, and deleted though views.
- UPDATAEBLE VIEWS are those in which views are used against INSERT, DELETE and UPDATE statements.

The following conditions must be fulfilled for view updates:

- DISTINCT must not be specified; that is, duplicate rows must not be eliminated from the query results.
- The FROM clause must specify only one updateable table; that is, the view must have a Single source table for which the user has the required privileges. If the source table is itself a view, then that view must meet these criteria.
- Each select item must be a simple column reference; the select list cannot contain expressions, calculated columns, or column functions.
- The WHERE clause must not include a subquery; only simple row-by-row search conditions may appear.
- The query must not include a GROUP BY or a HAVING clause.

The following code block has an example to update the age of Ramesh.

```
UPDATE CUSTOMERS_VIEW
  SET AGE = 35
  WHERE name = 'Ramesh';
```

This would ultimately update the base table CUSTOMERS and the same would reflect in the view itself.

10.7 DCLSTATEMENTS: -

10.7.1 Introduction to Database privileges:

When multiple users can access database objects, authorization can be controlled to these objects with privileges. Every object has an owner. Privileges control if a user can modify an object owned by another user. Privileges are granted or revoked either by the instance administrator, a user with the ADMIN privilege or, for privileges to a certain object, by the owner of the object.

1) System Privileges:

System privileges are privileges given to users to allow them to perform certain functions that deal with managing the database and the server
e.g Create user, Create table, Drop table etc.

2) Object Privileges:

Object privileges are privileges given to users as rights and restrictions to change contents of database object – where database objects are things like tables, stored procedures, indexes, etc.
Ex. Select,insert,delete,update,execute,references etc

Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges.

10.7.2 DCL defines two commands: -

- **Grant:** Gives user access privileges to database.
- **Revoke:** Take back permissions from user.

Procedure for granting privileges:

Grant: This command is used to give permission to user to dooperations on the other user's object.
Syntax:
Grant<object privileges>**on**<objectname>**to**<username>[with grant option] ;
Example:
GRANTSELECT, UPDATE ON student FROM BCA, MCA;

Procedure for revoking privileges:

Revoke: This command is used to withdraw the privileges that has been

granted to a user.

Syntax:
Revoke<object privileges>**on**<object name>**from**<username> ;

Example:
REVOKE SELECT, UPDATE ON student FROM BCA, MCA;

10.7.3 Viewing Privileges:

- ✓ **To Allow a User to create Session**

grant create session to username;

- ✓ **To Allow a User to create Table**
  grant create table to username;

- ✓ **To provide User with some Space on Tablespace to store Table**
  alter user username quota unlimited on system;

- ✓ **To Grant all privilege to a User**
  Grantsysdba to username

- ✓ **To Grant permission to Create any Table**
  grant create any table to username

- ✓ **To Grant permission to Drop any Table**
  grant drop any table to username

- ✓ **To take back Permissions**
  revoke create table from username

10.8 LIST OF REFERENCES

www.w3school.com
www.tutorialspoint.com
wielyIndia.com or DreamtechPress.com
www.williamstannings.com

10.9 BIBLIOGRAPHY

Database System Concepts(4th Edition) by Korth, Tata McGraw Hill

Introduction to Database Management Systems by ISRD Group, Tata McGraw Hill

SQL ,PL/SQL the Programming language of Oracle by Ivan Bayross(4th edition), BPB.

Advanced Database Management System by Dasgupta Chakrabarti, Dreamtech

10.10 EXERCISES

1) Describe concept of subqueries with example.
2) Describe system and object privileges and also describe use of Grant and Revoke commands with suitable example.
3) What are the main tasks performed by DBA?
4) Explain Role and Responsibilities of DBA?
5) What is OUTER JOIN? Explain in detail.
6) With the help of example, explain DROP VIEW command.
7) What are the views? Give syntax and example of creating view.
8) Explain string, data and math functions of SQL.