CHAPTER 1

Introduction and Applications of DBMS (Database Management System):

A Database Management System (DBMS) is software that facilitates the creation, management, and manipulation of databases. It provides an organized and efficient way to store, retrieve, update, and delete data. The main goal of a DBMS is to provide data security, data integrity, data consistency, and data concurrency.

Applications of DBMS:

1. Enterprise Resource Planning (ERP): Integrates various business processes like finance, human resources, inventory, and sales into a unified system.

2. Customer Relationship Management (CRM): Helps businesses manage interactions with customers, track leads, and improve customer satisfaction.

3. Online Transaction Processing (OLTP): Handles real-time transactional processing, e.g., online shopping, banking transactions.

4. Data Warehousing: Stores historical data for analysis and decision-making processes.

5. Healthcare Information Systems: Manages patient records, medical data, and billing information.

6. Social Media Platforms: Store and retrieve user-generated content, profiles, and connections.

File Processing System and its Limitations:

In the File Processing System (FPS), data is organized into files, and each application program has its own files. It lacks centralization and standardized access methods. Some limitations of FPS include:

1. Data Redundancy: The same data may be duplicated in multiple files, leading to inefficiency and inconsistency.

2. Data Inconsistency: Updates to data in one file may not be propagated to other related files, causing discrepancies.

3. Lack of Data Integrity: FPS lacks mechanisms to enforce data integrity rules, resulting in the possibility of invalid data.

4. Difficulty in Data Access and Sharing: Different programs may have different file formats and structures, making it challenging to share data.

5. No Data Independence: Changes in the structure of files may require modifying all programs using those files.

ANSI/SPARC Model:

The ANSI/SPARC (American National Standards Institute/Standards Planning and Requirements Committee) model is a conceptual framework for database management systems. It consists of three levels:

1. External Schema (User View): Describes the user's view of the database and represents subsets of the database relevant to specific user groups.

2. Conceptual Schema (Logical View): Represents the entire database using a high-level, abstract view, independent of physical storage details.

3. Internal Schema (Physical View): Describes the physical storage and implementation details of the database on the underlying hardware.

Data Independence:

Data Independence refers to the ability to modify the database schema at one level without affecting the schemas at the higher levels. There are two types of data independence:

1. Logical Data Independence: The ability to change the conceptual schema without impacting the external schemas or applications that use the data.

2. Physical Data Independence: The ability to modify the internal schema without affecting the conceptual and external schemas.

Client-Server Architecture:

Client-Server Architecture is a distributed computing model in which tasks are divided between "clients" (user interfaces) and "servers" (data processing and storage). Clients send requests to servers, and servers process those requests and return the results.

Example: Consider a web application where users (clients) interact with a web browser to access and use services hosted on a web server (server). The web server handles requests for web pages, processes application logic, and interacts with a database server to retrieve or update data.

Users & DBA (Database Administrator):

1. Users: Users interact with the database to perform various operations like querying data, inserting new records, updating existing data, and deleting records. Users can be classified into end-users, application programmers, and database administrators.

2. Database Administrator (DBA): The DBA is responsible for the overall management of the database system. Their tasks include database design, security management, performance optimization, backup and recovery, and ensuring data integrity and availability.

Database Architecture:

Database architecture refers to the overall structure and design of a database management system. It can be divided into three main components:

1. Internal Level (Physical Storage): This level deals with how data is physically stored on the storage devices, such as hard disks.

2. Conceptual Level (Logical Structure): This level represents the logical structure of the entire database, including tables, relationships, and constraints.

3. External Level (User Views): This level defines the various user views, which provide a customized and relevant perspective of the data to different user groups.

Example: Suppose a retail company has a database system. At the internal level, the database architecture deals with how the data is physically stored on servers. At the conceptual level, it includes the data model and entity-relationship diagram representing the entire retail database. At the external level, there could be different views for employees, managers, and customers, each providing relevant information tailored to their needs.

CHAPTER 2 –

SQL (Structured Query Language):

SQL is a standard language used to interact with databases. It allows users to define, manipulate, and control data in a relational database management system (RDBMS). SQL consists of several components, including Data Definition Language (DDL) commands, Data Manipulation Language (DML) commands, Data Control Language (DCL) commands, and Transaction Control Language (TCL) commands. Additionally, SQL supports various predicates, logical operators, relational operators, and different types of functions.

1. Data Definition Language (DDL) commands:

DDL commands are used to define and manage the structure of database objects, such as tables, indexes, and constraints.

Examples:

- CREATE TABLE: Creates a new table in the database.

```
  sql
CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    emp_age INT
);
```

- ALTER TABLE: Modifies the structure of an existing table.

```
  sql
ALTER TABLE Employees
ADD emp_salary DECIMAL(10, 2);
```

- DROP TABLE: Deletes an existing table from the database.

```sql
DROP TABLE Employees;
```

2. Data Manipulation Language (DML) commands:

DML commands are used to manipulate data within the database tables.

Examples:

- INSERT: Adds new records into a table.

```sql
INSERT INTO Employees (emp_id, emp_name, emp_age)
VALUES (1, 'John Doe', 30);
```

- UPDATE: Modifies existing records in a table.

```sql
UPDATE Employees
SET emp_age = 31
WHERE emp_id = 1;
```

- DELETE: Removes records from a table.

```sql
DELETE FROM Employees
WHERE emp_id = 1;
```

3. Data Control Language (DCL) commands:

DCL commands are used to control access permissions and privileges in the database.

Examples:

- GRANT: Provides specific privileges to database users.

  sql
```
GRANT SELECT, INSERT ON Employees TO john_user;
```

- REVOKE: Removes specific privileges from database users.

  sql
```
REVOKE INSERT ON Employees FROM john_user;
```

4. Transaction Control Language (TCL) commands:

TCL commands are used to manage transactions in the database.

Examples:

- COMMIT: Saves all the changes made during a transaction.

  sql
```
COMMIT;
```

- ROLLBACK: Discards all the changes made during a transaction.

  sql
```
ROLLBACK;
```

5. Predicates & Clauses:

Predicates are used in SQL queries to filter and retrieve specific data from the database. Clauses are used to define conditions or constraints in SQL statements.

Examples:

- Logical Operators (AND / OR):

```sql
SELECT * FROM Employees
WHERE emp_age > 25 AND emp_salary > 50000;
```

- Relational Operators:

```sql
SELECT * FROM Employees
WHERE emp_age >= 30;
```

- BETWEEN Predicate:

```sql
SELECT * FROM Employees
WHERE emp_age BETWEEN 25 AND 35;
```

- IN & NOT IN Predicate:

```sql
SELECT * FROM Employees
WHERE emp_id IN (1, 2, 3);
```

- LIKE Predicate:

```sql
SELECT * FROM Employees
WHERE emp_name LIKE 'John%';
```

6. Functions in SQL:

SQL provides various functions to perform operations on data.

Examples:

- Aggregate Functions:

```sql
SELECT COUNT(*) AS TotalEmployees FROM Employees;
SELECT SUM(emp_salary) AS TotalSalary FROM Employees;
```

- Character Functions:

```sql
SELECT UPPER(emp_name) AS UppercaseName FROM Employees;
SELECT CONCAT(emp_name, ' is ', emp_age, ' years old.') AS
Description FROM Employees;
```

- Arithmetic Functions:

```sql
SELECT emp_salary * 1.1 AS IncreasedSalary FROM Employees;
```

- Date Functions:

  sql

```
SELECT CURRENT_DATE() AS Today;
SELECT DATE_ADD(CURRENT_DATE(), INTERVAL 7 DAY) AS NextWeek;
```

- Conversion Functions:

  sql

```
SELECT CAST(emp_age AS CHAR) AS AgeString FROM Employees;
```

These notes provide an overview of SQL, covering DDL, DML, DCL, and TCL commands, as well as various predicates, clauses, and functions with examples. SQL is a powerful language that enables efficient data management and manipulation in relational databases.

CHAPTER 3 –

Data Models:

1. Hierarchical Model:

- The Hierarchical data in a tree-like structure where each record has a single parent and can have multiple child records.
- Example: An organizational chart where each employee has one manager and may have several subordinates.

2. Network Model:

- The Network Model allows each record to have multiple parent and child records, creating a more complex interconnected structure.

- Example: A university database where a course can have multiple prerequisites and a student can take multiple courses.

3. Relational Model:

- The Relational Model represents data in tabular form, with rows as records and columns as attributes.

- Example: A simple table for storing student information with columns like ID, Name, Age, and GPA.

4. Object-Oriented Model:

- The Object-Oriented Model combines data and methods (functions) into objects, providing better representation for real-world entities.

- Example: A class "Car" with attributes like make, model, and year, along with methods like "start_engine" and "accelerate."

E-R Diagram (Entity-Relationship Diagram):

Introduction to E-R Diagram:

- An E-R Diagram is a graphical representation used to model the structure of a database.

- It visualizes the entities, their attributes, and the relationships between entities.

Entities:

- Entities are real-world objects or concepts represented in the database.

- Example: In a university database, "Student" and "Course" are entities.

Attributes & its Types:

- Attributes are properties or characteristics of an entity.

- Example: In the "Student" entity, attributes can be "Student ID," "Name," "Age," and "Major."

- Attribute Types:

  - Simple attribute: Atomic values (e.g., Name, Age).

  - Composite attribute: Composed of multiple sub-attributes (e.g., Address with sub-attributes Street, City, Zip).

  - Single-valued attribute: Contains a single value (e.g., Age).

  - Multi-valued attribute: Contains multiple values (e.g., Phone Numbers).

  - Derived attribute: Calculated or derived from other attributes (e.g., Age calculated from Birthdate).

Relationships:

- Relationships represent associations between entities.

- Example: A "Student" entity may have a relationship "Enrolls" with the "Course" entity.

Mapping Cardinalities:

- Mapping cardinalities define the number of instances of an entity that can be associated with another entity through a relationship.

- Examples: One-to-One (1:1), One-to-Many (1:N), Many-to-One (N:1), Many-to-Many (N:N).

Participation Constraints:

- Participation constraints define whether an entity's participation in a relationship is mandatory or optional.

- Examples: Total Participation (Mandatory) and Partial Participation (Optional).

Weak Entity Sets:

- Weak Entity Sets are entities that depend on another entity for identification.

- They have a partial key and require a relationship with a strong entity for identification.

- Example: In a database for bank transactions, a "Transaction" entity may depend on the "Account" entity for identification.

Specialization and Generalization:

- Specialization is the process of dividing an entity into sub-entities based on specific attributes or characteristics.

- Generalization is the process of combining sub-entities back into a single entity.

- Example: A "Person" entity can be specialized into "Student" and "Employee" sub-entities.

Aggregation:

- Aggregation represents a relationship where one entity is composed of multiple smaller entities.

- Example: In a hospital database, an "Inpatient" entity can aggregate multiple "Medical Record" entities.

E-R Diagrams are powerful tools for database design as they provide a clear visual representation of the database structure, entities, attributes, relationships, and constraints. They help in understanding the database schema and assist in database development and maintenance.

CHAPTER 4 –

Relational Data Model:

Introduction:

- The Relational Data Model represents data in the form of tables (relations) consisting of rows (tuples) and columns (attributes).

- It establishes relationships between tables using keys and allows for efficient data retrieval and manipulation.

Degree:

- Degree refers to the number of attributes (columns) in a relation (table).

- Example: In a "Student" table with attributes like ID, Name, and Age, the degree is 3.

Cardinality:

- Cardinality refers to the number of tuples (rows) in a relation (table).

- Example: If a "Student" table contains 100 records, the cardinality is 100.

Constraints & Keys:

Primary Key:

- A Primary Key uniquely identifies each tuple (row) in a relation.

- Example: In a "Student" table, the "ID" attribute can be set as the primary key.

Foreign Key:

- A Foreign Key is an attribute in one relation that refers to the Primary Key in another relation.

- It establishes relationships between tables.

- Example: In an "Enrollments" table, the "StudentID" attribute may be a foreign key referencing the "ID" attribute in   "Student" table.

Super Key:

- A Super Key is a set of one or more attributes that can uniquely identify a tuple (row) in a relation.

- It may contain extra attributes compared to the minimum required for a Primary Key.

- Example: In a "Student" table, the combination of "ID" and "Email" attributes forms a super key.

Candidate Key:

- A Candidate Key is a minimal super key, meaning it is a set of attributes that uniquely identifies a tuple without any redundant attributes.

- Example: In a "Student" table, the "ID" attribute is a candidate key.

Not Null Constraint:

- The Not Null Constraint ensures that an attribute cannot have a null (missing) value.

- Example: In a "Product" table, the "Price" attribute may have a Not Null Constraint to ensure all products have a defined price.

Check Constraint:

- The Check Constraint sets a condition that must be met for an attribute's value to be valid.

- Example: In an "Employee" table, the "Age" attribute may have a Check Constraint to ensure it is a positive integer.

Relational Algebra Operations:

Selection:

- The Selection operation retrieves rows from a relation that satisfy a specified condition (predicate).

- Example: Select all students with an age greater than 20 from the "Student" table.

Projection:

- The Projection operation retrieves specific attributes (columns) from a relation while discarding others.
- Example: Retrieve the "Name" and "GPA" attributes from the "Student" table.

Cross-Product:

- The Cross-Product operation combines each tuple from one relation with every tuple from another relation, creating a new relation.
- Example: Combine each student from the "Student" table with each course from the "Course" table.

Rename:

- The Rename operation changes the names of attributes in a relation without altering the data.
- Example: Change the attribute name "ID" to "StudentID" in the "Student" table.

Joins (Natural & Outer Join):

- Joins combine tuples from two relations based on a specified condition.
- Natural Join combines tuples with matching values in attributes with the same name.
- Outer Join includes unmatched tuples from one or both relations.
- Example: Perform a Natural Join between the "Student" and "Enrollments" tables.

Set Operators (Union, Intersection, Set Difference):

- Set Operators combine tuples from two relations based on set theory principles.
- Union combines tuples from both relations, excluding duplicates.
- Intersection includes only common tuples from both relations.
- Set Difference includes tuples present in one relation but not the other.

- Example: Perform a Union operation between two "Student" tables to combine all unique student records.

Aggregate Functions:

- Aggregate Functions perform calculations on groups of tuples and return single values.
- Examples: SUM, AVG, COUNT, MAX, MIN.
- Example: Calculate the average GPA of all students in the "Student" table.

Relational Data Model, along with its constraints, keys, and algebra operations, forms the foundation for the design, implementation, and manipulation of relational databases. Understanding these concepts is essential for effective database management and querying.

CHAPTER 5 –

Relational Database Design:

Functional Dependency (FD):

- Functional Dependency is a relationship between two attributes in a relation, where the value of one attribute uniquely determines the value of another.
- It is represented as X -> Y, where X determines Y.
- Example: In a "Student" table with attributes "StudentID" and "Name," if StudentID -> Name, it means each StudentID uniquely determines a student's Name.

Trivial and Non-Trivial FD:

- Trivial FD: An FD where Y is a subset of X, meaning X -> Y is true by default.

- Example: If the "Student" table has an attribute "Age," then StudentID, Age -> Age is a trivial FD.

- Non-Trivial FD: An FD where Y is not a subset of X, and it provides new information.

- Example: StudentID -> Name is a non-trivial FD.

Armstrong's Axioms/Inference Rules:

Armstrong's Axioms are a set of inference rules used to derive all possible functional dependencies in a relation.

1. Reflexivity: If X is a set of attributes, then X -> X.
2. Augmentation: If X -> Y, then XZ -> YZ for any Z.
3. Transitivity: If X -> Y and Y -> Z, then X -> Z.

Closure of FD:

- The Closure of FD (F+) is the set of all functional dependencies that can be inferred from a given set of FDs.

- Example: If we have FDs StudentID -> Name and Name -> GPA, then the closure is StudentID -> Name, GPA.

Closure of Attributes:

- The Closure of Attributes (X+) is the set of all attributes functionally determined by a given set of attributes X.

- Example: If we have FDs A -> B, B -> C, then A+ is A, B, C.

Candidate Key:

- A Candidate Key is a set of attributes that can uniquely identify each tuple in a relation.

- It should have the property that removing any attribute from the set will result in a loss of the uniqueness property.

- Example: In a "Student" table, {StudentID} is a candidate key as it uniquely identifies each student.

Finding a Candidate Key:

- To find candidate keys, we identify attribute sets with no redundant attributes and have the unique identification property.

- Example: In the "Student" table, {StudentID} and {Email} are candidate keys.

Decomposition (Lossy & Lossless):

- Decomposition is the process of breaking a relation into two or more smaller relations to eliminate redundancy.

- Lossy Decomposition: Some information may be lost after the decomposition, leading to data inconsistencies.

- Lossless Decomposition: No information is lost after the decomposition, and the original data can be reconstructed.

- Example of Lossy Decomposition: Decomposing a "Student" table into "Personal Info" and "Academic Info" tables may lead to data anomalies if the link between them is lost.

- Example of Lossless Decomposition: Decomposing a "Student" table into "Student ID" and "Student Details" tables, where "Student ID" is the candidate key, is lossless.

Database Anomalies:

- Database Anomalies occur when data inconsistencies arise due to redundancy or incorrect normalization.

- Types of Anomalies: Insertion Anomalies, Deletion Anomalies, Update Anomalies.

Normalization:

1. First Normal Form (1NF):

- Eliminate repeating groups and ensure each attribute contains only atomic values.

- Example: Splitting a "Course" attribute containing multiple course names into separate rows.

2. Second Normal Form (2NF):

   - Satisfy 1NF and remove partial dependencies.

   - Example: Create separate tables for student details and course enrollment to eliminate partial dependencies.

3. Third Normal Form (3NF):

   - Satisfy 2NF and remove transitive dependencies.

   - Example: If a "Department Head" attribute is functionally dependent on "Department," create a separate "Department" table with its head.

4. Boyce-Codd Normal Form (BCNF):

   - Satisfy 3NF and remove overlapping candidate keys.

   - Example: If a relation has two candidate keys, ensure each non-key attribute is fully dependent on all candidate keys.

5. Fourth Normal Form (4NF):

   - Satisfy BCNF and remove multi-valued dependencies.

   - Example: Create separate tables for attributes with multi-valued dependencies.

6. Fifth Normal Form (5NF):

   - Satisfy 4NF and remove join dependencies.

   - Example: Decompose a relation containing join dependencies into separate tables.

Normalization helps to avoid data redundancies, inconsistencies, and anomalies, leading to better database design and improved data integrity.