1)Mention the needs for standardization of DBMS Software.

Ans: **Needs for Standardization of DBMS Software:**

1. **Interoperability:** Enables different database systems to work together and exchange data smoothly.
2. **Portability:** Applications can be easily migrated between different DBMS platforms with minimal changes.
3. **Reduced Learning Curve:** Standard languages like SQL make it easier for users to learn and use any DBMS.
4. **Vendor Independence:** Prevents lock-in with a single vendor, offering flexibility in choosing or switching DBMS providers.
5. **Improved Tool Support and Security:** Ensures broader compatibility with tools and promotes uniform security practices.

2. a)Discuss the roles a Data Base Administrator(DBA) in DBMS.

b)Discuss the role of Conceptual Schema in the Three-level Architecture in DBMS.

Ans: **Roles of a Database Administrator (DBA):**

The **DBA** is responsible for managing and maintaining the database system. Key roles include:

1. **Database Design** – Helps in designing the database schema and structure.
2. **Performance Monitoring** – Optimizes query performance and system response time.
3. **Backup and Recovery** – Ensures regular backups and restores data after failures.
4. **Security Management** – Controls user access, permissions, and data security.
5. **Storage Management** – Manages disk space and database storage requirements.
6. **User Management** – Creates user accounts and assigns appropriate roles.

b) Role of Conceptual Schema in Three-Level Architecture:

In the **three-level architecture** (external, conceptual, internal), the **conceptual schema** plays the central role:

1. **Definition**: It represents the **entire logical view** of the database, independent of physical storage.

2. **Purpose**: Acts as a bridge between the external views (user-specific) and the internal schema (physical).
3. **Central Database Structure**: Defines all entities, relationships, constraints, and data types.
4. **Data Independence**: Helps achieve logical data independence, allowing changes in the logical structure without affecting user views.

3. Describe the different types of database users with respect to their roles.

Ans:

User Type	**Role / Description**
Database Administrator (DBA)	Manages the database system, handles design, security, backup, recovery, and performance.
Application Programmers	Develop applications that interact with the database using programming languages and SQL.
End Users	Access and use data; categorized into:
– Casual Users	Occasionally query the database using a query language.
– Naive Users	Use pre-designed interfaces or forms (e.g., ATM users).
– Sophisticated Users	Perform complex queries and data analysis using advanced tools.
– Stand-alone Users	Use off-the-shelf applications for personal or small-scale use.
System Analysts/Designers	Analyze user needs and design the database structure and applications.

4.Mention the roles of different types of database users with respect to the architecture of a RDBMS package.

Ans:

User Type	**Role in RDBMS Architecture**
Database Administrator (DBA)	Operates at the **internal level**; manages physical storage, indexing, security, and performance tuning.

User Type	Role in RDBMS Architecture
Application Programmers	Work at the **conceptual level**; design application logic and write programs to interact with the database.
End Users	Interact with the **external level**; use views and interfaces provided by applications or queries.
– Casual Users	Use ad-hoc queries to interact with data.
– Naive Users	Access data through predefined forms and interfaces without understanding database structure.
– Sophisticated Users	Use advanced tools for data mining, querying, and reporting.
– Stand-alone Users	Use independent applications with a fixed schema and interface.
System Analysts/Designers	Bridge between user requirements and system design; align external and conceptual levels.

5.Justify the statement “Data is an important asset for decision making in a business”.

Ans: **Justification:**

1. **Informed Decisions**: Data provides factual evidence that helps managers make informed and objective decisions rather than relying on guesswork.
2. **Understanding Trends**: Analyzing historical data reveals patterns and trends, allowing businesses to forecast sales, customer behavior, and market changes.
3. **Improving Efficiency**: Operational data helps identify bottlenecks and inefficiencies, leading to optimized processes and cost reduction.
4. **Customer Insights**: Data about customer preferences, feedback, and behavior enables personalized services, enhancing customer satisfaction and loyalty.
5. **Competitive Advantage**: Businesses that effectively analyze and use data can respond faster to market demands and outperform competitors.

6. a)Define data integrity in the context on DBMS.

b)Distinguish between physical data integrity and logical data integrity.

Ans:a) **Data Integrity** refers to the **accuracy, consistency, and reliability** of data stored in a database. It ensures that the data remains **correct and valid** throughout its lifecycle, preventing unauthorized changes, duplication, or corruption.

b)

Aspect	Physical Data Integrity	Logical Data Integrity
Definition	Ensures data is stored **accurately and safely** on physical media.	Ensures data is **logically correct, consistent, and meaningful**.
Focus	Protects data from hardware failures, corruption, or power loss.	Maintains correct relationships, formats, and business rules.
Example	Backups, RAID, checksums to avoid physical data loss.	Enforcing foreign keys, constraints, and data validation rules.
Layer in DBMS	Lower (storage and hardware level).	Higher (schema and application level).

7.Mention with examples the various components of an ER Diagram.

Component	Description	Example
Entity	Objects or things in the real world represented as rectangles.	Student, Course, Employee
Attribute	Properties of entities, shown as ovals connected to entities.	Student_ID, Name, DOB
Key Attribute	An attribute that uniquely identifies an entity, underlined in the diagram.	Student_ID, Employee_ID
Composite Attribute	Attributes composed of multiple sub-parts.	Name → First_Name, Last_Name
Derived Attribute	Attribute that can be derived from another, shown with a dashed oval.	Age (from DOB)
Multivalued Attribute	Attribute with multiple values, shown with a double oval.	Phone_Numbers
Relationship	Associations between entities, shown as diamonds.	Enrolled, Works_For
Cardinality	Specifies the number of entity instances involved in a relationship.	One-to-One, One-to-Many
Participation	Indicates whether all entities are involved (total) or some (partial).	All Students must enroll → total participation

8. i) What is an ER-Diagram? Mention its use in Database design. I

i) Draw an ER-diagram for a University Academic Management system considering different types of database users, different kinds of roles & all possible data storages. Clearly, mentions the assumptions made.

Ans: An **ER-Diagram (Entity-Relationship Diagram)** is a **visual representation** of entities (real-world objects), their **attributes**, and **relationships** between them in a database.

Uses in Database Design:

- Helps in **conceptual database design** before physical implementation.
- Clarifies **business rules** and data requirements.
- Facilitates communication between **developers, designers, and stakeholders**.
- Acts as a blueprint for converting to relational schemas.

---

ii) Assumptions Made:

- Users include **Admin, Faculty, Students**, and **Staff**.
- Roles include **Course Registration, Exam Management, Attendance Tracking, Result Publishing**, etc.
- Data storage involves **Course Details, Student Records, Faculty Info, Exam Schedules, Results**.

Entity	Attributes
Student	Student_ID (PK), Name, DOB, Email, Course_ID (FK)
Faculty	Faculty_ID (PK), Name, Department, Email
Course	Course_ID (PK), Title, Credits, Department
Exam	Exam_ID (PK), Course_ID (FK), Date, Type
Result	Result_ID (PK), Student_ID (FK), Exam_ID (FK), Marks
User	User_ID (PK), Role, Login_ID, Password
Department	Dept_ID (PK), Dept_Name

Relationships:

- Student **enrolls** in Course (Many-to-Many)
- Faculty **teaches** Course (One-to-Many)
- Student **appears** in Exam (Many-to-Many)
- Exam **belongs to** Course (Many-to-One)
- Result **is for** Exam and Student (Many-to-One)
- User **has** Role (One-to-Many)
- Department **offers** Course (One-to-Many)

9. Explain the meaning of "Data integrity". Distinguish between physical data integrity and logical data integrity.

Ans:

Aspect	**Physical Data Integrity**	**Logical Data Integrity**
Definition	Ensures data is **physically stored** correctly and protected from **hardware issues**, corruption, and loss.	Ensures the **logical correctness** of data by enforcing business rules and relationships between entities.
Focus	Focuses on the **storage** medium (e.g., hard drives, network storage).	Focuses on the **data structure**, relationships, and rules within the database schema.
Examples	Backups, RAID, checksums to prevent data corruption due to hardware failure.	Enforcing **foreign key constraints**, **unique keys**, and **check constraints** to ensure data consistency.
Layer in DBMS	Operates at the **lower level** (storage and physical layer).	Operates at the **higher level** (schema and logical structure layer).
Methods	Use of **redundancy**, **data recovery tools**, and **hardware maintenance**.	Use of **validation rules**, **referential integrity**, and **data normalization**.

10. Define the different kinds of integrity constraints. Associate them with different types of keys in a database management system. Also, mention the benefits of the integrity constraints.

Ans:

Integrity Constraint	Definition	Associated Key	Example
Domain Integrity	Ensures data in a column is of the **correct type** and **within valid values**.	No direct association	"Salary" column must contain **positive numbers**, "DOB" column must contain **valid dates**.
Entity Integrity	Ensures every record has a **unique identifier**, and primary key is **not NULL**.	Primary Key (PK)	"Student_ID" in "Student" table is the unique identifier; cannot be NULL.
Referential Integrity	Ensures foreign keys refer to **valid records** in the referenced table.	Foreign Key (FK)	"Student_ID" in "Enrollment" table must refer to a valid "Student_ID" in the "Student" table.
Key Integrity	Ensures values in candidate key columns are **unique** for each record.	Candidate Key	"Employee_ID" and "SSN" are candidate keys in the "Employee" table, both must be unique.
Check Integrity	Ensures values in a column meet **specific conditions or criteria**.	No direct association	"Age" column must contain values between **18 and 100**.

11. Mention the needs for standardization of DBMS Software. Explain the role of Codd's Rules in this context.

Ans: **Needs for Standardization of DBMS Software**

1. **Interoperability**: Ensures seamless communication between different database systems.
2. **Portability**: Allows databases to be easily migrated across platforms.
3. **Consistency**: Guarantees uniform database operations across systems.
4. **Vendor Independence**: Reduces reliance on specific vendors.
5. **Easy Maintenance**: Simplifies upgrades and system management.

Role of Codd's Rules

1. **Uniform Data Representation**: Ensures data is stored in tables with consistent formats.

2. **Data Integrity**: Promotes rules like unique identifiers and null value handling.
3. **Standard Query Language**: Ensures use of relational languages like SQL for uniform querying.
4. **Data Independence**: Guarantees changes in physical or logical data don't affect the user interface.
5. **Enforces Constraints**: Ensures integrity constraints like foreign keys and primary keys are maintained.

12. Illustrate the role of Conceptual Schema in the Three-level Architecture in DBMS.

Ans: **Data Independence**: The conceptual schema provides **logical data independence** by isolating users from the physical storage details (internal schema) and external views.

 Unified View of Data: It offers a **global view** of the entire database, capturing the logical structure and relationships between data without worrying about physical storage.

 Ensures Consistency: It maintains **data integrity** and ensures consistency across various user views (external schemas) by defining common data models and constraints.

 Facilitates Data Management: The conceptual schema allows for the **definition of data types**, constraints, and relationships, which are essential for database management and querying.

 Separation of Concerns: It separates the concerns of **how** the data is stored (internal schema) from **what** data is stored and how it relates (conceptual schema).

13. Write down the steps for conversion of an ER-Diagram into a Relational Database with proper explanation. Consider all possible cases such as Hierarchical Entity Sets, Composites Attributes, Weak Entity Sets etc.

Ans:

ER Concept	**Relational Database Equivalent**
Entity	Table (Primary Key)
Relationship	Foreign Key (1:1, 1:N) or Junction Table (M:N)
Attribute	Table Column
Composite Attribute	Separate Columns for each component

ER Concept	Relational Database Equivalent
Multi-valued Attribute	New Table with Foreign Key referencing original table
Weak Entity	Table with Foreign Key from Parent (Strong Entity)
Generalization/Specialization	Superclass and Subclass Tables (with Foreign Keys)

14. Justify the need for having different types of keys while designing a database

Ans:

Type of Key	Purpose	Need/Justification	Example
Primary Key (PK)	Ensures each record is unique and identifiable.	- Guarantees **uniqueness** of records.- **No NULL values** allowed.- Ensures **data integrity**.	Student_ID in a **Student** table to uniquely identify each student.
Foreign Key (FK)	Links a field in one table to the primary key of another table.	- Ensures **referential integrity**.- Prevents **orphaned records**.- Defines **relationships** between tables.	Student_ID in the **Enrollment** table linking to Student_ID in the **Student** table.
Candidate Key	A set of attributes that can uniquely identify a record in a table.	- Provides **flexibility** to choose the most appropriate key.- Ensures **uniqueness**.	Email or Phone_Number can uniquely identify a **Customer**.
Composite Key	A combination of multiple attributes to uniquely identify a record.	- Useful for **complex data** where a single attribute isn't enough.- Ensures **unique combination**.	Combination of Order_ID and Product_ID in an **OrderDetails** table.
Superkey	Any set of attributes that can uniquely identify a record.	- Ensures **uniqueness**.- Broadest set of attributes but not necessarily minimal.	Combination of Student_ID, Name, and DOB as a superkey.
Unique Key	Ensures all values in a column are unique, but allows **NULL values**.	- **Allows optional uniqueness**.- Permits **NULLs** in certain columns.	Username in a **User** table, allowing unique entries but Middle_Name can be NULL

15. Justify the statement, “Logical data independence is more difficult to achieve than physical data independence.”

Ans: **Logical Data Independence** and **Physical Data Independence** both aim to reduce the impact of changes in a database, but **logical data independence** is harder to achieve for the following reasons:

1. **Nature of Changes**:
    - **Physical Changes** (e.g., storage techniques) can be managed easily without affecting the logical schema or user views.
    - **Logical Changes** (e.g., altering tables, adding/removing attributes) directly impact user views, applications, and queries, making them more challenging.
2. **User Impact**:
    - **Physical Independence**: Users are unaware of changes in storage and can continue using the system without modification.
    - **Logical Independence**: Users must adapt to changes, as they affect how data is structured and queried.
3. **Complexity of Implementation**:
    - **Physical Independence** is relatively easy to implement using DBMS features like indexing and storage management.
    - **Logical Independence** requires significant changes to views, queries, and applications, making it more complex to achieve.
4. **Performance Considerations**:
    - **Physical Changes** do not usually affect system performance.
    - **Logical Changes** can impact performance and may require query optimization and redesign.

16. Considering two tables Emp(Empno, Name, Sal, Deptno) and Dept(Deptno, Dname), write a query to find out the departments having no employees.

=> SELECT d.Deptno, d.Dname
FROM Dept d
LEFT JOIN Emp e ON d.Deptno = e.Deptno
WHERE e.Empno IS NULL;

17. Illustrate Self Join with an example.

=>

18. Simple Join vs Natural Join
    =>

Feature	Simple Join (INNER JOIN)	Natural Join
Join Condition	Requires **explicit** `ON` clause to specify matching columns.	Automatically joins using **columns with the same name** in both tables.
Column Selection	You manually define the columns to join on.	Database automatically detects and joins on common column names.
Flexibility	More flexible	Less flexible
Control	Full control over which columns to match.	Less control
Portability	Supported by all databases.	Not supported by all databases
Preferred When	You want **custom logic or full control**.	You want **quick and automatic join** based on same-named columns.
example	SELECT Emp.Name, Dept.Dname FROM Emp INNER JOIN Dept ON Emp.DeptID = Dept.DeptID;	SELECT Name, Dname FROM Emp NATURAL JOIN Dept;

---

19. Justify the need of having a standard Query Language for making a business query from a database of an organization.
    =>
    Justification for the Need of a Standard Query Language in Business Queries

```
1.Uniformity: Ensures consistent communication with databases across    teams, simplifying training and collaboration.

2.Ease of Use: Enables efficient data retrieval and manipulation with simple, powerful syntax.
```

```
3.Platform Independence: Works across various relational databases, reducing the need to rewrite queries.

4.Better Decision-Making: Allows quick access to accurate data for informed business strategies.

5.Security: Supports role-based access to protect sensitive information.

2. Automation & Integration: Easily integrates with apps and tools for automated reporting and workflows.

7.Scalability: Standardized queries are easier to maintain and optimize as data grows.
```

---

20. Describe the different Tools (ways) through which a database user can make a Query on a database.

=> **Tools for Querying a Database**

1. **Command-Line Interface (CLI) Tools**
   Tools like MySQL CLI, psql (for PostgreSQL), or SQL*Plus (for Oracle) allow users to write and execute SQL commands directly in a terminal.
2. **Graphical User Interfaces (GUIs)**
   Applications like **phpMyAdmin**, **DBeaver**, **HeidiSQL**, and **SQL Server Management Studio (SSMS)** provide user-friendly interfaces to run queries, manage databases, and view results visually.
3. **Database Management Systems (DBMS)**
   Software like **Oracle**, **MySQL**, **PostgreSQL**, and **Microsoft SQL Server** have built-in query editors that let users write and execute SQL queries.
4. **Web-Based Interfaces**
   Many modern applications provide web dashboards that allow users to run custom or predefined queries (e.g., in admin panels or BI tools like Metabase).
5. **Programming Languages**
   Users can write database queries through code using languages like **Python (with SQLAlchemy or SQLite3)**, **Java (JDBC)**, **PHP**, **Node.js**, etc.
6. **Business Intelligence (BI) Tools**
   Tools like **Power BI**, **Tableau**, and **Looker** allow users to build and run queries through visual drag-and-drop interfaces, often generating SQL in the background.
7. **APIs and Web Services**
   Applications can send queries to databases indirectly using **RESTful APIs** or **GraphQL**, especially in multi-tiered web applications.

---

21. Compare Relational Algebra with Relational Calculus citing proper examples

=>

Feature	Relational Algebra	Relational Calculus
Type	**Procedural** (specifies *how* to obtain result)	**Non-Procedural** (specifies *what* result is desired)
Operations Used	Uses operators like **σ** (select), **π** (project), ∪, **−**, **×**, ⋈ (join)	Uses logical predicates and quantifiers (∃, ∀)
Ease of Use	More intuitive for programmers	More mathematical and abstract
Example in Relational Algebra	π_name (σ_dept_name='Sales' (Employee ⋈ Department))	{ s.name | Student(s) ∧ s.grade = 'A' }
Focus	Sequence of operations to retrieve data	Logical description of desired data
Implementation	Easier to implement in query processors	Requires translation to algebra for implementation

23. Considering two tables Emp(Empno, Name, Sal, Deptno) and Dept(Deptno, Dname), write a query to find out the departments having no employees. Categorize the type of join used in this case.

=> SELECT D.Deptno, D.Dname
FROM Dept D
LEFT JOIN Emp E ON D.Deptno = E.Deptno
WHERE E.Deptno IS NULL;

Type of join used :

✅ **Left Outer Join**

This join includes all records from the left table (`Dept`), and the matching records from the right table (`Emp`). If there is no match, `NULL` is returned for columns from the right table — which we use to find departments without employees.

24. Considering two tables Emp(Empno, Name, Sal, Deptno) and Dept(Deptno, Dname), write a query to find out the departments having no employees.

=> SELECT D.Deptno, D.Dname
FROM Dept D
WHERE D.Deptno NOT IN (SELECT E.Deptno FROM Emp E);

OR

```
SELECT Dept.Deptno, Dept.Dname
FROM Dept
```

```
LEFT JOIN Emp ON Dept.Deptno = Emp.Deptno
WHERE Emp.Deptno IS NULL;
```

25. Compare Inner Join and Outer Join with an example.
    =>

🔄 Comparison Table:

Feature	Inner Join	Outer Join
Definition	Returns **only matching rows** between tables	Returns **matching + non-matching rows**
Types	Only one: Inner Join	Left Outer, Right Outer, and Full Outer Join
Null Rows	Rows with no match are **excluded**	Rows with no match are **included with NULLs**
Use Case	When you want data common to both tables	When you want all data from one or both tables
Example	SELECT Emp.Name, Dept.DeptName FROM Emp INNER JOIN Dept ON Emp.DeptID = Dept.DeptID;	SELECT Emp.Name, Dept.DeptName FROM Emp LEFT JOIN Dept ON Emp.DeptID = Dept.DeptID;

26. . Cite All Relational Algebra Operations with proper examples. Compare Relational Algebra with Relational Calculus with respect to their functional aspects.
    =>

🔹 Relational Algebra Operations (with Examples)

1. **Selection (σ)** – Selects rows based on condition
   Example:
   `σ Salary > 50000 (Employee)` → Employees with salary > ₹50,000
2. **Projection (π)** – Selects specific columns
   Example:
   `π Name, Salary (Employee)` → Names and salaries of employees
3. **Union (∪)** – Combines rows from two relations
   Example:
   `π EmpID (Employee) ∪ π EmpID (Manager)` → All employee and manager IDs
4. **Set Difference (−)** – Rows in one relation not in another
   Example:
   `π EmpID (Employee) − π EmpID (Manager)` → Employees who are not managers
5. **Cartesian Product (×)** – Combines all rows of two relations
   Example:
   `Employee × Department` → All combinations of employee and department
6. **Rename (ρ)** – Renames relation or attributes
   Example:

`ρ Temp(EmpID, EName, DeptID, Salary)(Employee)` → Renames `Employee` to `Temp`

7. **Join (⋈)** – Combines rows with matching attributes
   Example:
   `Employee ⋈ Employee.DeptID = Department.DeptID` → Employees with their departments
8. **Intersection (∩)** – Rows common to both relations
   Example:
   `π EmpID (Employee) ∩ π EmpID (Manager)` → Employees who are also managers
9. **Division (÷)** – Used for "for all" type queries
   Example:
   `π EmpID (WorksOn) ÷ π DeptID (Department)` → Employees working in all departments

🔹 Relational Algebra vs Relational Calculus

Feature	Relational Algebra	Relational Calculus
Type	Procedural (describes **how** to retrieve)	Non-Procedural (describes **what** to retrieve)
Basis	Based on set theory and operations	Based on predicate logic
Usage	Easy for implementation in DBMS	Good for expressing complex queries logically
Query Form	Sequence of operations	Logical expressions using variables

27. Compare DDL, DML & DCL Statements. List the different Application areas of SQL

=>

Feature	DDL (Data Definition Language)	DML (Data Manipulation Language)	DCL (Data Control Language)
Purpose	Defines and modifies database structure	Used to manipulate data in the database	Controls access to data
Operations	Create, Alter, Drop, Truncate	Insert, Update, Delete, Select	Grant, Revoke
Affects	Structure (schema)	Data	Permissions and access
Rollback Possible	❌ No	✅ Yes	✅ Yes
Example	`CREATE TABLE Student(...)`	`INSERT INTO Student VALUES (...)`	`GRANT SELECT ON Student TO user1`

🔸 Application Areas of SQL

1. **Database Management Systems** (e.g., MySQL, Oracle, SQL Server)
2. **Data Analysis & Reporting** (via queries, aggregations, and filtering)
3. **Web and App Development** (backend integration with databases)

4. **Business Intelligence Tools** (e.g., Power BI, Tableau use SQL for data extraction)
5. **Data Warehousing** (for querying and organizing large datasets)

---

28. Differentiate Tuple Relational Calculus from Domain Relational Calculus.
    =>

Difference Between Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC)

Basis of Comparison	Tuple Relational Calculus (TRC)	Domain Relational Calculus (DRC)
Definition	The Tuple Relational Calculus (TRC) is used to select tuples from a relation. The tuples with specific range values, tuples with certain attribute values, and so on can be selected.	The Domain Relational Calculus (DRC) employs a list of attributes from which to choose based on the condition. It's similar to TRC, but instead of selecting entire tuples, it selects attributes.
Tuple/ Domain	A tuple is a single element of relation. In database terms, it is a row.	A domain is equivalent to column data type and any constraints on the value of data.
Filtering	This filtering variable uses a tuple of relations.	This filtering is done based on the domain of attributes.
Similarity	It reflects traditional pre-relational file structures.	It is more similar to logic as a modeling language.
Focus	Focuses on selecting tuples from a relation	Focuses on selecting values from a relation
Ease of use	Easier to use for simple queries.	More difficult to use for simple queries.
Use case	Useful for selecting tuples that satisfy a certain condition or for retrieving a subset of a relation.	Useful for selecting specific values or for constructing more complex queries that involve multiple relations.

29. (Answer)

=> (a) Create another table named `employees1` containing all records of the `employees` table:

```
CREATE TABLE employees1 AS SELECT * FROM employees;
```

(b) Create another table named `employees2` containing only the structure and no records of the `employees` table:

```
CREATE TABLE employees2 AS SELECT * FROM employees WHERE 1=0;
```

(c) Select the name of the person having no manager:

```
SELECT first_name, last_name FROM employees WHERE manager_id IS NULL;
```

30. Describe the various Relational Algebra Operations with Suitable Examples.

=>

1. Selection (σ)

Selection filters rows from a relation based on a condition.

Notation: $\sigma_{condition}(Relation)$

Example: Consider a `Students` relation with attributes (ID, Name, Age, Major):

```
σ(Age > 20)(Students)
```

This operation selects all students older than 20 years.

2. Projection (π)

Projection extracts specific columns from a relation.

Notation: $\pi_{attributes}(Relation)$

Example:

```
π(Name, Major)(Students)
```

This returns only the Name and Major columns from the Students relation.

3. Union (∪)

Union combines two relations with the same schema.

Notation: Relation1 ∪ Relation2

Example: If we have `CSStudents` and `MathStudents` relations:

```
CSStudents ∪ MathStudents
```

This gives all students from both CS and Math departments (without duplicates).

4. Set Difference (−)

Set difference returns tuples in the first relation that are not in the second.

Notation: Relation1 − Relation2

Example:

```
Students - GraduatedStudents
```

This returns students who haven't graduated yet.

5. Cartesian Product (×)

Cartesian product combines every tuple from the first relation with every tuple from the second.

Notation: Relation1 × Relation2

Example: With `Students` and `Courses` relations:

```
Students × Courses
```

This creates all possible combinations of students and courses.

6. Rename (ρ)

Rename changes the name of a relation or its attributes.

Notation: $\rho_{NewName}$(Relation) or $\rho_{NewName(A1,A2,...)}$(Relation)

Example:

```
ρ(Undergrads)(σ(Level='UG')(Students))
```

31/33.Write SQL Statements for the following cases.

a) **Add a new tuple in the Customers table with suitable data.**

INSERT INTO Customers (CustomerID, CustomerName, ContactName, Address, City, PostalCode, Country)

VALUES (101, 'Tech Solutions', 'Alice Johnson', '123 Elm Street', 'New York', '10001', 'USA');

b) **Create a backup table named Cust_BackUp containing all records of "Customers" table.**

CREATE TABLE Cust_BackUp AS

SELECT * FROM Customers;

c) **Delete all records from "Customers" table for all customers who live in the city of "Berlin" or the Country of "UK".**

DELETE FROM Customers

WHERE City = 'Berlin' OR Country = 'UK';

d) **Show the Customer names and Addresses for all customers who do not live in the Countries of "UK" or "Mexico" in a sorted way in Alphabetic order of their Cities.**

SELECT CustomerName, Address

FROM Customers

WHERE Country NOT IN ('UK', 'Mexico')

ORDER BY City ASC;

e) **Show the count of Distinct Country names in the customers table.**

SELECT COUNT(DISTINCT Country) AS DistinctCountryCount

FROM Customers;

32. Justify the need of having a standard Query Language for making a business query from a database of an organization. Describe the different Application areas of SQL in this context.

A **standard query language** like **SQL (Structured Query Language)** is essential in any organization for the following reasons:

1.**Uniformity & Standardization**: SQL provides a **universal language** that all relational database systems understand. It ensures consistency across teams and platforms.

2.**Ease of Use**: SQL is relatively easy to learn and read, making it accessible for business analysts and non-developers.

3.**Data Manipulation**: SQL allows users to **retrieve, insert, update, and delete** data easily using queries.

4.**Decision Making**: Businesses rely heavily on data insights. SQL enables **complex queries** for generating reports, KPIs, summaries, and business intelligence.

5. **Security**: SQL supports **access control** via roles and permissions, ensuring only authorized access to sensitive data.

6. **Integration**: SQL integrates well with **front-end apps, dashboards, and reporting tools** like Power BI, Tableau, etc.

34. i)Write SQL Queries for creating all of the tables for the above depicted Banking System maintaining all of the relationships and necessary constraints.

```
CREATE TABLE Bank (
   Code VARCHAR(10) PRIMARY KEY,
   Name VARCHAR(100),
   Address VARCHAR(200)
);

CREATE TABLE Branch (
   Branch_id VARCHAR(10) PRIMARY KEY,
   Name VARCHAR(100),
   Address VARCHAR(200),
   Bank_Code VARCHAR(10),
   FOREIGN KEY (Bank_Code) REFERENCES Bank(Code)
);

CREATE TABLE Loan (
   Loan_id VARCHAR(10) PRIMARY KEY,
   Loan_type VARCHAR(50),
   Amount DECIMAL(15,2),
   Branch_id VARCHAR(10),
   FOREIGN KEY (Branch_id) REFERENCES Branch(Branch_id)
);
```

```
CREATE TABLE Account (
    Account_No VARCHAR(20) PRIMARY KEY,
    Acc_Type VARCHAR(50),
    Balance DECIMAL(15,2),
    Branch_id VARCHAR(10),
    FOREIGN KEY (Branch_id) REFERENCES Branch(Branch_id)
);

CREATE TABLE Customer (
    Custid VARCHAR(10) PRIMARY KEY,
    Name VARCHAR(100),
    Phone VARCHAR(15),
    Address VARCHAR(200)
);

CREATE TABLE Loan_Customer (
    Loan_id VARCHAR(10),
    Custid VARCHAR(10),
    PRIMARY KEY (Loan_id, Custid),
    FOREIGN KEY (Loan_id) REFERENCES Loan(Loan_id),
    FOREIGN KEY (Custid) REFERENCES Customer(Custid)
);

CREATE TABLE Account_Customer (
    Account_No VARCHAR(20),
    Custid VARCHAR(10),
    PRIMARY KEY (Account_No, Custid),
    FOREIGN KEY (Account_No) REFERENCES Account(Account_No),
    FOREIGN KEY (Custid) REFERENCES Customer(Custid)
);
```

ii)Mention the Normal Forms for each of the tables so created with proper justification.

Normal Forms of Each Table

1. Bank

- **1NF**: Atomic values (no repeating groups)
- **2NF**: No partial dependency (Code is PK)
- **3NF**: No transitive dependencies

✅ **Normal Form: 3NF**

---

2. Branch

- **1NF**: Atomic attributes
- **2NF**: Fully functionally dependent on PK (Branch_id)
- **3NF**: No transitive dependency

✅ **Normal Form: 3NF**

---

3. Loan

- **1NF**: Atomic
- **2NF**: Fully functionally dependent on PK (Loan_id)
- **3NF**: No transitive dependency

✅ **Normal Form: 3NF**

---

4. Account

- **1NF**: Atomic
- **2NF**: Fully functionally dependent on PK (Account_No)
- **3NF**: No transitive dependency

✅ **Normal Form: 3NF**

---

5. Customer

- **1NF**: Atomic
- **2NF**: Fully functionally dependent on PK (Custid)
- **3NF**: No transitive dependency

✅ **Normal Form: 3NF**

---

6. Loan_Customer

- **1NF**: Atomic
- **2NF**: Composite PK (Loan_id, Custid) and fully dependent
- **3NF**: No transitive dependency

✅ **Normal Form: 3NF**

---

7. Account_Customer

- Same reasoning as Loan_Customer

✅ **Normal Form: 3NF**

35.Justify the advantages of BCNF in database normalization

BCNF (Boyce-Codd Normal Form) Advantages:

- **Eliminates Redundancy**: Ensures that **every non-trivial functional dependency** has a determinant that is a **candidate key**, removing duplicate data.
- **Avoids Anomalies**: Eliminates **insertion, deletion, and update anomalies**.
- **Improved Consistency**: Reduces data inconsistency and ensures better **data integrity**.
- **Efficient Storage**: Optimizes the use of memory by minimizing data repetition.
- **Better Schema Design**: Leads to logically clean and well-structured schema.

36. Roles played by Functional Dependencies in database design

✅ **Functional Dependencies (FDs):**

A **Functional Dependency** occurs when one attribute uniquely determines another.

Roles:

- **Foundation of Normalization**: FDs are the **basis for normal forms**.
- **Schema Decomposition**: Help decide how to **break tables** into smaller ones.
- **Data Integrity**: Ensures that **relationships between attributes** are maintained.
- **Identification of Keys**: Help find **candidate and primary keys**.

37/38(i). Define Normalization. Enlist its advantages

Definition:

Normalization is the process of organizing data in a database to reduce **data redundancy** and improve **data integrity** by dividing tables and establishing relationships.

Advantages:

- Removes **duplicate data**
- Prevents **anomalies** (insertion, update, deletion)
- Enhances **data consistency**
- Improves **query performance**
- Ensures **data integrity**
- Facilitates **database maintenance**

38(ii). Compare Partial and Transitive Dependency with concrete examples.

Dependency Type	Definition	Example
Partial Dependency	When a non-prime attribute is dependent on part of a composite key	In a table Student(CourseID, StudentID, StudentName), if StudentName depends only on StudentID
Transitive Dependency	When a non-prime attribute depends on another non-prime attribute	In Student(ID, DeptID, DeptName), DeptName depends on DeptID, not on the key ID directly

- Partial Dependencies are removed in 2NF.
- Transitive Dependencies are removed in 3NF.

Example:

A table where the **primary key is composite**.

Table: StudentCourse(StudentID, CourseID, StudentName, CourseName)

StudentID	CourseID	StudentName	CourseName
101	C101	Alice	DBMS
102	C102	Bob	OS

🔍 **Explanation:**

- Composite primary key: (StudentID, CourseID)
- StudentName depends **only on StudentID**, not the full key.
- So, **StudentName is partially dependent** on the primary key.

This is a Partial Dependency. It violates 2NF.
To fix it, we split the table into:

- Student(StudentID, StudentName)
- Enrollment(StudentID, CourseID)
- Course(CourseID, CourseName)

Transitive Dependency Example (violates 3NF)

Scenario:

A table where a **non-key attribute depends on another non-key attribute**.

Table: Student(StudentID, StudentName, DeptID, DeptName)

StudentID	StudentName	DeptID	DeptName
101	Alice	D01	Computer Science
102	Bob	D02	Mechanical

🔍 **Explanation:**

- Primary key: StudentID
- DeptName depends on DeptID, and DeptID depends on StudentID.
- So, **DeptName is transitively dependent** on StudentID.

This is a Transitive Dependency. It violates 3NF.
To fix it, we split the table into:

- Student(StudentID, StudentName, DeptID)
- Department(DeptID, DeptName)

39. Explain Insertion Anomalies with example.

Insertion anomalies are problems that occur when you cannot insert data into a database because some required information is missing or because the insertion would create inconsistent data. These anomalies typically occur in poorly designed databases that haven't been properly normalized.

Example Scenario

Let's consider a poorly designed database for a university system with a single table:

```
CREATE TABLE Student_Course (
    StudentID INT,
```

```
    StudentName VARCHAR(100),
    CourseID INT,
    CourseName VARCHAR(100),
    Instructor VARCHAR(100),
    PRIMARY KEY (StudentID, CourseID)
);
```

1. Simple Insertion Anomaly (Cannot add a new course without students)

Problem: You can't add information about a new course until at least one student enrolls in it

```
INSERT INTO Student_Course (CourseID, CourseName, Instructor)
VALUES (101, 'Database Systems', 'Dr. Smith');
```

Error: This would fail because StudentID is part of the primary key and cannot be null.

2. Referential Insertion Anomaly (Cannot add a student without courses)

Problem: You can't add a new student until they enroll in at least one course.

```
INSERT INTO Student_Course (StudentID, StudentName)
VALUES (1001, 'John Doe');
```

Error: This would fail because CourseID is part of the primary key and cannot be null.

3. Data Redundancy and Potential Inconsistency

Even when you can insert data, the design forces redundancy:

```
INSERT INTO Student_Course VALUES (1001, 'John Doe', 101, 'Database Systems', 'Dr. Smith');
INSERT INTO Student_Course VALUES (1001, 'John Doe', 102, 'Algorithms', 'Dr. Johnson');
INSERT INTO Student_Course VALUES (1002, 'Jane Smith', 101, 'Database Systems', 'Dr. Smith');
```

Notice how:

- "John Doe" appears twice
- "Database Systems" appears twice with the same instructor
- Any change to course information would need to be made in multiple places

40. Explain Updation Anomalies with proper example.

An **update anomaly** occurs when updating data in a poorly structured table leads to inconsistencies because the same data is stored in multiple places. This happens in denormalized tables where data redundancy exists.

Example of Update Anomaly

Poorly Designed Table (Denormalized)

Consider a single table storing **Student-Course** information:

sql

Copy

Download

```
CREATE TABLE Student_Course (
   StudentID INT,
   StudentName VARCHAR(100),
   CourseID INT,
   CourseName VARCHAR(100),
   Instructor VARCHAR(100),
   PRIMARY KEY (StudentID, CourseID)
);
```

Problem Scenario

Suppose **Dr. Smith** is the instructor for **CourseID 101 (Database Systems)**, and multiple students are enrolled in this course:

sql

Copy

Download

```
INSERT INTO Student_Course VALUES
(1001, 'John Doe', 101, 'Database Systems', 'Dr. Smith'),
(1002, 'Jane Smith', 101, 'Database Systems', 'Dr. Smith'),
(1003, 'Alice Brown', 101, 'Database Systems', 'Dr. Smith');
```

Now, if **Dr. Smith** leaves and is replaced by **Dr. Johnson**, we must update **all rows** where CourseID = 101:

sql

Copy

Download

```
UPDATE Student_Course
SET Instructor = 'Dr. Johnson'
WHERE CourseID = 101;
```

Issue:

- If we miss updating even one row, the data becomes **inconsistent**.
- Some records may still show **Dr. Smith**, while others show **Dr. Johnson**.

41. Explain Deletion Anomalies with proper example.

A **deletion anomaly** occurs when deleting data unintentionally removes other important information due to poor database design.

Example of Deletion Anomaly

Using the Same Denormalized Table

sql

Copy

Download

```
CREATE TABLE Student_Course (
   StudentID INT,
   StudentName VARCHAR(100),
   CourseID INT,
   CourseName VARCHAR(100),
   Instructor VARCHAR(100),
   PRIMARY KEY (StudentID, CourseID)
);
```

Problem Scenario

Suppose **John Doe (StudentID 1001)** is the **only student** enrolled in **CourseID 102 (Algorithms)**:

sql

Copy

Download

```
INSERT INTO Student_Course VALUES
(1001, 'John Doe', 101, 'Database Systems', 'Dr. Smith'),
(1001, 'John Doe', 102, 'Algorithms', 'Dr. Lee');
```

If **John Doe drops CourseID 102**, we delete his enrollment:

sql

Copy

Download

```
DELETE FROM Student_Course
WHERE StudentID = 1001 AND CourseID = 102;
```

Issue:

- The entire record for **"Algorithms" (CourseID 102)** is lost because it was only stored with John Doe's enrollment.
- Now, there is **no record** of this course in the database, even if it still exists.

42. Define BCNF. Compare BCNF and 3NF.

A relation is in **Boyce-Codd Normal Form (BCNF)** if:

For every **non-trivial functional dependency (X → Y)**, **X is a super key**.

3NF vs BCNF:

Feature	**3NF (Third Normal Form)**	**BCNF**
Condition	Every non-prime attribute is non-transitively dependent on candidate key	Every determinant is a candidate key
Handles anomalies	Removes most anomalies	Removes **all anomalies**, including those missed in 3NF
Dependency on non-superkey	**Allowed** if right side is a prime attribute	**Not allowed**
Strictness	Less strict	More strict
Example where 3NF fails	A table with FD: CourseID → TeacherName and TeacherName → Dept	In BCNF, both FDs must have determinants that are candidate keys

43. Define 3NF. Check if the Relation (R) = {A, B, C, D, E} with FDs = {AB -> CDE, E -> B} (in 1NF) is in 3NF or not. Justify your answer and if not then decompose R to 3NF.

A relation is in **3NF** if:

1. It is in **2NF**, and
2. For every **non-trivial functional dependency** X → Y, **at least one** of the following holds:

- X is a **super key**, or
- Y is a **prime attribute** (part of a candidate key)

- elation **R = {A, B, C, D, E}**
- FDs:
  1. **AB → CDE**
  2. **E → B**

We assume R is already in **1NF** (no repeating groups).

Step 1: Find candidate keys

We need to find a minimal set of attributes that can determine all attributes in R.

- **AB → CDE** → Closure of AB = ABCDE → So, **AB is a candidate key**
- Any smaller subset?
  - A^+ = A → not enough
  - B^+ = B → not enough
  - E^+ = B (from E→B), but not all attributes
  - AE^+ = A, E, B (E→B), but no C or D
  - So, **AB is the only candidate key**

Prime attributes: **A and B** (since they're part of a candidate key)

Step 2: Check each FD against 3NF

1. **AB → CDE**
   - LHS is a candidate key → Satisfies 3NF
2. **E → B**
   - LHS E is **not** a super key
   - RHS B is a **prime attribute**
   - Satisfies 3NF condition 2

Conclusion: The relation R is already in 3NF — because all FDs satisfy at least one of the 3NF conditions.

44. Explain 4th Normal Form.

Definition:

A relation is in **4NF** if:

- It is in **Boyce-Codd Normal Form (BCNF)**
- It has **no multi-valued dependencies (MVDs)**

Multi-Valued Dependency (MVD):

Occurs when, for a single attribute value, there are **multiple independent values** of two or more other attributes.

Example:

Table: Student(Course, Hobby)

Course	**Hobby**
CS	Painting
CS	Music

Here, a student can take **multiple courses** and have **multiple hobbies**, **independent** of each other. This leads to redundancy.

✅ In 4NF, we separate the MVDs into different relations:

- StudentCourses(Student, Course)
- StudentHobbies(Student, Hobby)

45. Explain 5th Normal Form.

A relation is in **5NF (Project-Join Normal Form)** if:

- It is in **4NF**, and
- Every **join dependency** is implied by the candidate keys of the relation.

Purpose:

To **eliminate redundancy** caused by **join dependencies**, not just FDs or MVDs.

Example:

Suppose a table stores:

Supplier	Part	Project
S1	P1	J1
S1	P2	J1
S1	P1	J2
S1	P2	J2

There may be **redundant combinations**, and to reconstruct the original table, we may need to **join multiple projections**.

In 5NF, the relation is decomposed in such a way that we avoid **spurious tuples** and **preserve lossless join**.

46. a) What do you mean by Functional Dependency? Explain their role in Normalization of Database.

A **Functional Dependency (FD)** is a constraint between two sets of attributes in a relation. Formally, for a relation **R**, a FD **X → Y** implies that if two tuples have the same value for **X**, they must also have the same value for **Y**. Here, **X** is called the determinant, and **Y** is the dependent attribute(s).

Example: In a student table, if `RollNo → Name`, it means for every RollNo, the Name is uniquely determined.

Role in Normalization:
FDs are the foundation of **normalization**, which is the process of organizing data to minimize redundancy and dependency. During normalization:

- FDs help identify **candidate keys**, **super keys**, and **prime attributes**.
- They are used to break large, redundant tables into smaller ones through various **normal forms** (1NF, 2NF, 3NF, BCNF).
- This reduces **insertion**, **update**, and **deletion anomalies**, ensuring better data integrity.

---

46. b) In context of Functional Dependency define Super Key, Candidate Key and Primary key.

- **Super Key**: A super key is any combination of attributes that uniquely identifies a tuple in a relation. It may contain unnecessary attributes (i.e., not minimal).
    - Example: In a student table, `{RollNo, Name}` can be a super key if RollNo is unique.
- **Candidate Key**: A candidate key is a **minimal** super key, meaning no attribute can be removed without losing the ability to uniquely identify a record.
    - If `{RollNo}` alone uniquely identifies students, then it's a candidate key.
- **Primary Key**: Among all candidate keys, one is selected as the primary key. This key is used to uniquely identify tuples and cannot contain NULL values.
    - Example: Choosing `RollNo` as the primary key over `{RollNo, Name}`.

All three are essential in designing a normalized and efficient schema.

---

47. Explain how we can find the Closure F^+ of a Set of Functional Dependencies represented by F.

The **closure F^+** of a set of functional dependencies **F** is the set of all functional dependencies that can be **logically inferred** from F using **Armstrong's Axioms**. This helps in:

- Checking whether a given dependency is implied by F.
- Finding candidate keys.

Steps to compute Closure of Attribute Set X (denoted X^+):

1. Start with $X^+ = X$.
2. Repeatedly apply FDs in F:
    - If $Y \rightarrow Z$ is in F and $Y \subseteq X^+$, then add Z to X^+.
3. Repeat until no more attributes can be added.

Armstrong's Axioms:

- **Reflexivity**: If $Y \subseteq X$, then $X \rightarrow Y$.
- **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$.
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Example:
$F = \{A \rightarrow B, B \rightarrow C\}$, find A^+:
Start: $A^+ = \{A\}$
Apply $A \rightarrow B$: $A^+ = \{A, B\}$
Apply $B \rightarrow C$: $A^+ = \{A, B, C\}$
Thus, $A^+ = \{A, B, C\}$

---

48. Define 3NF. Check if the Relation (R) = {A, B, C, D, E} with FDs = {AB → CDE, E → B} (in 1NF) is in 3NF or not. Justify your answer and if not then decompose R to 3NF.

Third Normal Form (3NF):
A relation is in 3NF if:

1. It is in 2NF.
2. For every FD $X \rightarrow Y$, at least one of the following holds:
    - $X \rightarrow Y$ is a trivial FD ($Y \subseteq X$).
    - X is a super key.
    - Every attribute in Y is a **prime attribute** (part of a candidate key).

Given:

- Relation R = {A, B, C, D, E}
- FDs: $AB \rightarrow CDE$, $E \rightarrow B$

Step 1: Find Candidate Key(s):
From $AB \rightarrow CDE \rightarrow R$, AB is a candidate key.

Step 2: Prime Attributes: A and B (from candidate key AB).

Check Each FD:

- **$AB \rightarrow CDE$**: AB is a candidate key → satisfies 3NF

- **E → B**: E is not a super key, and B is a **prime attribute**, so it satisfies the 3rd condition of 3NF

Answer: The relation is in **3NF** as all dependencies satisfy 3NF conditions.

---

49. Define Indexing in DBMS. Mention the different kinds of indexing techniques for a DBMS. Explain its role for fast retrieval of data from a Relational Table.

Indexing is a data structure technique that improves the speed of data retrieval operations on a database table. Indexes are similar to book indexes—they help locate information quickly without scanning every record.

Role in Fast Retrieval:

- Reduces disk I/O.
- Helps in locating rows faster based on key values.
- Improves performance of **SELECT** queries, **JOINs**, and **ORDER BY** operations.

Types of Indexing:

1. **Primary Index**:
    - Based on the primary key.
    - Entries are in the same order as the physical data.
2. **Secondary Index**:
    - Used for non-primary key attributes.
    - May not follow the physical data order.
3. **Clustered Index**:
    - Sorts and stores table rows in order of the index key.
    - Only one per table.
4. **Non-Clustered Index**:
    - Contains pointers to the actual data.
    - Multiple can be created on a table.
5. **Dense Index**:
    - Contains an index record for every search key value.
6. **Sparse Index**:
    - Contains entries for only some records.
7. **Multilevel Index**:
    - Uses a hierarchy of indexes to reduce search time.
8. **B-Tree and B+ Tree Indexes**:
    - Self-balancing tree structures for efficient insertion, deletion, and search.

Conclusion: Indexing significantly boosts query performance by reducing the number of records to scan.

---

50. Define 2NF. Check if the Relation (R) = {A, B, C, D, E} with FDs = {AB → C, D → E} (in 1NF) is in 2NF or not. Justify your answer and if not then decompose R to 2NF.

Second Normal Form (2NF):
A relation is in 2NF if:

1. It is in 1NF.
2. It has **no partial dependency**, i.e., no non-prime attribute is functionally dependent on part of a candidate key.

Given:

- R = {A, B, C, D, E}
- FDs: AB → C, D → E

Step 1: Candidate Key: AB
So, **prime attributes** = A, B

Check for Partial Dependencies:

- AB → C: Full dependency → OK
- D → E: D is not a part of any key → not a partial dependency → OK

Conclusion: There are no partial dependencies.
The relation **is already in 2NF**.

51. For the Relational Schema R(a, b, c, d, e) with F = {a → c, b → ce, e → a, d → ab}

a) Find the Closure of {ab}, i.e., $\{ab\}^+$

To find the closure of {ab}, apply the given functional dependencies step-by-step:

Start: $\mathbf{\{ab\}^+ = \{a, b\}}$

Apply FD:

- a → c → add c → {a, b, c}
- b → ce → add c, e → {a, b, c, e}
- e → a → a already present → no change
- d → ab → not applicable (we don't have d)

→ Final closure: $\mathbf{\{ab\}^+ = \{a, b, c, e\}}$

---

b) Find the Super Key(s) of R

A **super key** is any attribute set whose closure contains all attributes {a, b, c, d, e}.
We need to find sets X such that $X^+ = R$.

From earlier:

- $ab^+ = \{a, b, c, e\} \rightarrow$ not full
- Try **d**: $d \rightarrow ab \rightarrow$ then $a \rightarrow c$, $b \rightarrow ce$, $e \rightarrow a$
  $\rightarrow d^+ = \{d, a, b, c, e\} \rightarrow$ all attributes

{d} is a super key
Any superset of {d} is also a super key.

---

c) Find the Candidate Key(s) of R

A **candidate key** is a **minimal super key**.

From above:

- d^+ gives all attributes → minimal set
  → **Candidate Key = {d}**

No subset of {d} gives all attributes, so it's minimal.

---

52. Explain the ACID Properties of a Transaction

ACID is an acronym for the four key properties that define a reliable transaction in a DBMS:

1. **Atomicity**
   - Ensures that a transaction is **all-or-nothing**.
   - If any part fails, the entire transaction is rolled back.
   - Example: Transferring money between two accounts—either both debit and credit happen or neither.
2. **Consistency**
   - Ensures that a transaction brings the database from one valid state to another.
   - All integrity constraints must be preserved.
   - No corruption of data is allowed.
3. **Isolation**
   - Transactions execute as if they are the **only** transaction in the system.
   - Prevents interference from concurrent transactions.
   - Isolation levels: Read Uncommitted, Read Committed, Repeatable Read, Serializable.
4. **Durability**
   - Once a transaction commits, changes are **permanent**, even in the event of a crash.
   - Ensured through logs and backups.

Together, ACID properties guarantee reliable and predictable database behavior, especially in multi-user environments.

---

53. Explain how concurrency is maintained in DBMS. Also, explain its advantages.

Concurrency control allows multiple transactions to execute simultaneously **without interfering** with each other, maintaining data consistency.

Techniques to Maintain Concurrency:

1. **Lock-based protocols** (e.g., 2-phase locking)
2. **Timestamp ordering**
3. **Optimistic concurrency control**
4. **Multiversion Concurrency Control (MVCC)**

Advantages of Concurrency:

- **Improved Throughput**: Multiple users can access the database at the same time.
- **Better Resource Utilization**: CPU and I/O devices are used more efficiently.
- **Data Consistency**: Ensures serializability of concurrent transactions.
- **Reduced Waiting Time**: Faster response times for users.

Concurrency control prevents problems like **dirty reads**, **lost updates**, **non-repeatable reads**, and **phantom reads**.

---

54. Explain 2 Phase Locking Protocol

Two-Phase Locking (2PL) is a concurrency control method that ensures **serializability** of transactions.

Phases:

1. **Growing Phase**:
    - Transaction acquires all the locks it needs.
    - No locks can be released in this phase.
2. **Shrinking Phase**:
    - Transaction releases locks.
    - No new locks can be acquired.

Types of Locks:

- **Shared Lock (S)**: For reading.
- **Exclusive Lock (X)**: For writing.

Advantages:

- Guarantees conflict-serializable schedules.
- Prevents inconsistent reads/writes.

Disadvantages:

- Can lead to **deadlocks**.
- Requires careful implementation in high-concurrency systems.

---

55. Describe the different recovery techniques used in Databases

Recovery techniques are needed to **restore the database** to a consistent state after failures such as system crashes, power loss, or transaction failures.

Main Techniques:

1. **Log-Based Recovery**
    - Maintains a **transaction log** (Write-Ahead Log - WAL).
    - Two types:
        - **Deferred Update**: Changes made after commit.
        - **Immediate Update**: Changes written during transaction but logged first.
2. **Checkpoints**
    - Periodically save database state.
    - During recovery, rollback only to the last checkpoint instead of the beginning.
3. **Shadow Paging**
    - Maintains **two copies** of the database.
    - Modifies shadow copy and swaps only after commit.
    - No need for logging.
4. **ARIES (Algorithm for Recovery and Isolation Exploiting Semantics)**
    - A sophisticated log-based recovery method.
    - Performs **analysis**, **redo**, and **undo** during recovery.
5. **Backup and Restore**
    - Regular backups + restore point.
    - Useful for catastrophic failures.

Objective: Minimize data loss and downtime.

56. What is Deadlock and how is it different from Starvation?

Deadlock is a situation where two or more transactions are waiting indefinitely for each other to release locks, and none of them can proceed. It occurs when:

- Transactions hold some resources (like locks) and wait for other resources held by others.
- A circular wait condition exists.

Example:

- T1 holds Lock A and waits for Lock B.
- T2 holds Lock B and waits for Lock A → **Deadlock**.

Starvation, on the other hand, happens when a transaction **waits indefinitely** because other higher-priority transactions are repeatedly given access to the resources it needs.

Differences:

Feature	Deadlock	Starvation
Cause	Circular waiting among transactions	Biased scheduling or priority
Resolution	Requires detection and recovery	Prevented using fair scheduling
Detection	Possible via wait-for graphs	Difficult to detect directly
Transactions Involved	All involved are blocked	Only lower-priority is delayed

Both problems need proper handling in concurrency control mechanisms.

---

57. Compare Indexing with Hashing in DBMS

Both **indexing** and **hashing** are used to speed up data retrieval in databases, but they differ in technique and use case.

Criteria	Indexing	Hashing
Method	Uses **B-trees**, B+ trees, etc.	Uses **hash functions**
Search Type	**Range-based and point** queries	**Exact match** queries only
Flexibility	Supports **ordered traversal**	No inherent order maintained
Efficiency	Better for **range and sorted** queries	Faster for **equality** comparisons
Data Structure	Tree-based	Hash tables
Example Use	Searching students in name order	Finding employee by ID

Conclusion:

- Use **indexing** when you need to perform range scans or sorting.
- Use **hashing** for constant-time access based on exact match queries.

---

58. Define Hashing in DBMS. Explain its role for fast retrieval of data from a Relational Table.

Hashing is a technique to map a search key to a location in the database using a **hash function**. It transforms a key into a hash value, which is used to quickly locate the record.

How it works:

- A **hash function** h(key) computes the address where the record is stored.
- Common hash functions: Division, Multiplication, Folding, Mid-square.

Types of Hashing:

1. **Static Hashing**:
    - Fixed number of buckets.
    - May cause overflow (chaining used).
2. **Dynamic Hashing**:
    - Grows and shrinks with data.
    - Techniques: Extendible Hashing, Linear Hashing.

Role in Fast Retrieval:

- Offers **constant time complexity (O(1))** for search.
- Especially efficient for **point queries** (e.g., find employee with ID 102).
- Reduces disk access by computing address directly.

Ideal for applications with large datasets and frequent key-based lookups.

---

59. Describe the different File Management Techniques with respect to Database Management

File management in DBMS deals with how data is stored and accessed on disk. There are various file organization techniques, each with its pros and cons.

1. Heap File (Unordered):

- Records are stored in the order they arrive.
- Easy to insert, but inefficient for search.

2. Sequential File:

- Records are stored in sorted order (e.g., by ID).
- Efficient for **range queries**, but insertion/deletion is costly.

3. Hashing File Organization:

- Uses hash function to determine record location.
- Fast for **exact match queries**, not good for range.

4. Clustered File Organization:

- Records with the same attribute values (like department) are stored together.
- Reduces I/O when accessing related records.

5. Indexed File Organization:

- Uses index files (like B-trees, B+ trees) to locate records quickly.
- Efficient for large databases and supports both exact and range queries.

DBMS handles these efficiently using a **buffer manager** to reduce disk I/O and ensure data consistency.

---

60. Explain the following terms:

a) Multi-valued Dependency (MVD):

- A type of dependency in which an attribute is **independent** of another in the context of a key.
- Notation: A →→ B means for a single A, multiple B values are allowed.
- Occurs when multiple values of one attribute depend on the same value of another attribute.
- Eliminated in **4NF**.

b) Join Dependency:

- A generalization of MVD.
- A relation R satisfies a join dependency if R = join of its projections.
- Eliminated in **5NF (Project-Join Normal Form)**.

c) Serializability of Transactions:

- Ensures that concurrent transactions result in a state that would be obtained if transactions were executed **serially**.
- Two types: **Conflict Serializability** and **View Serializability**.

d) Role of ER Diagram in DBMS:

- **Entity-Relationship (ER) diagrams** model the logical structure of databases.
- Show entities, attributes, relationships.
- Used in database design to translate real-world concepts into database schema.

e) B-Tree and B+ Tree:

- **B-Tree**: Balanced search tree used for indexing; all keys appear in internal and leaf nodes.
- **B+ Tree**: Variant of B-tree where all data is in leaf nodes and leaves are linked → Better for **range queries**.

f) Logical vs. Physical Integrity in DBMS:

- **Logical Integrity**: Ensures correctness of data (e.g., domain constraints, referential integrity).

- **Physical Integrity**: Protects against hardware failures, ensures data isn't corrupted due to system crashes.

---

61. In context of Three-Tier Architecture of DBMS, describe the Data Independence of DBMS. Explain with a clear diagram.

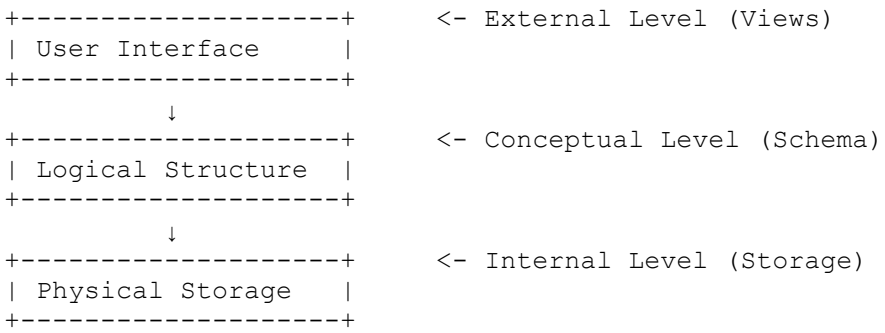
Three-Tier Architecture separates the database system into 3 layers:

1. **External Level (View Layer):**
    - User views.
    - Different users can see different data representations.
2. **Conceptual Level (Logical Layer):**
    - Logical structure of the entire database.
    - Defines entities, data types, relationships, constraints.
3. **Internal Level (Physical Layer):**
    - Actual storage and data structures used on disk.

Data Independence:

1. **Logical Data Independence**:
    - Ability to change **conceptual schema** without affecting external views or applications.
    - E.g., Adding a new attribute to a table.
2. **Physical Data Independence**:
    - Ability to change **internal schema** without changing the conceptual schema.
    - E.g., Changing file organization or storage devices.

Diagram:



This separation enhances **maintainability**, **scalability**, and **security**.

---

62. Explain the role of a DBA for RDBMS

A **Database Administrator (DBA)** is responsible for managing and maintaining the **Relational Database Management System (RDBMS)**. The DBA ensures data is available, consistent, secure, and properly backed up.

Roles and Responsibilities:

1. **Database Design**:
    - Help design schema, tables, constraints, and relationships.
    - Normalize data to reduce redundancy.
2. **User Access Control**:
    - Manage permissions.
    - Create users and assign roles to enforce **data security**.
3. **Performance Tuning**:
    - Optimize queries.
    - Create indexes and monitor slow-running operations.
4. **Backup and Recovery**:
    - Schedule regular backups.
    - Restore data in case of failure.
5. **Monitoring and Maintenance**:
    - Ensure uptime and handle logs.
    - Monitor resource usage and manage storage.
6. **Implementing Security Measures**:
    - Protect against unauthorized access.
    - Implement encryption, auditing, and intrusion detection.
7. **Update and Patch Management**:
    - Apply database patches and updates to keep the RDBMS up-to-date.

In summary, a DBA is vital for the **reliable, secure, and efficient operation** of a database system.