END TERM PAPER 2024 SOLUTION DBMD (SIMPLIFIED)

Question 1 (Compulsory)
(Attempt all parts - 3x5 = 15 Marks)

a) Describe the components of database systems. (3 Marks)

Answer:

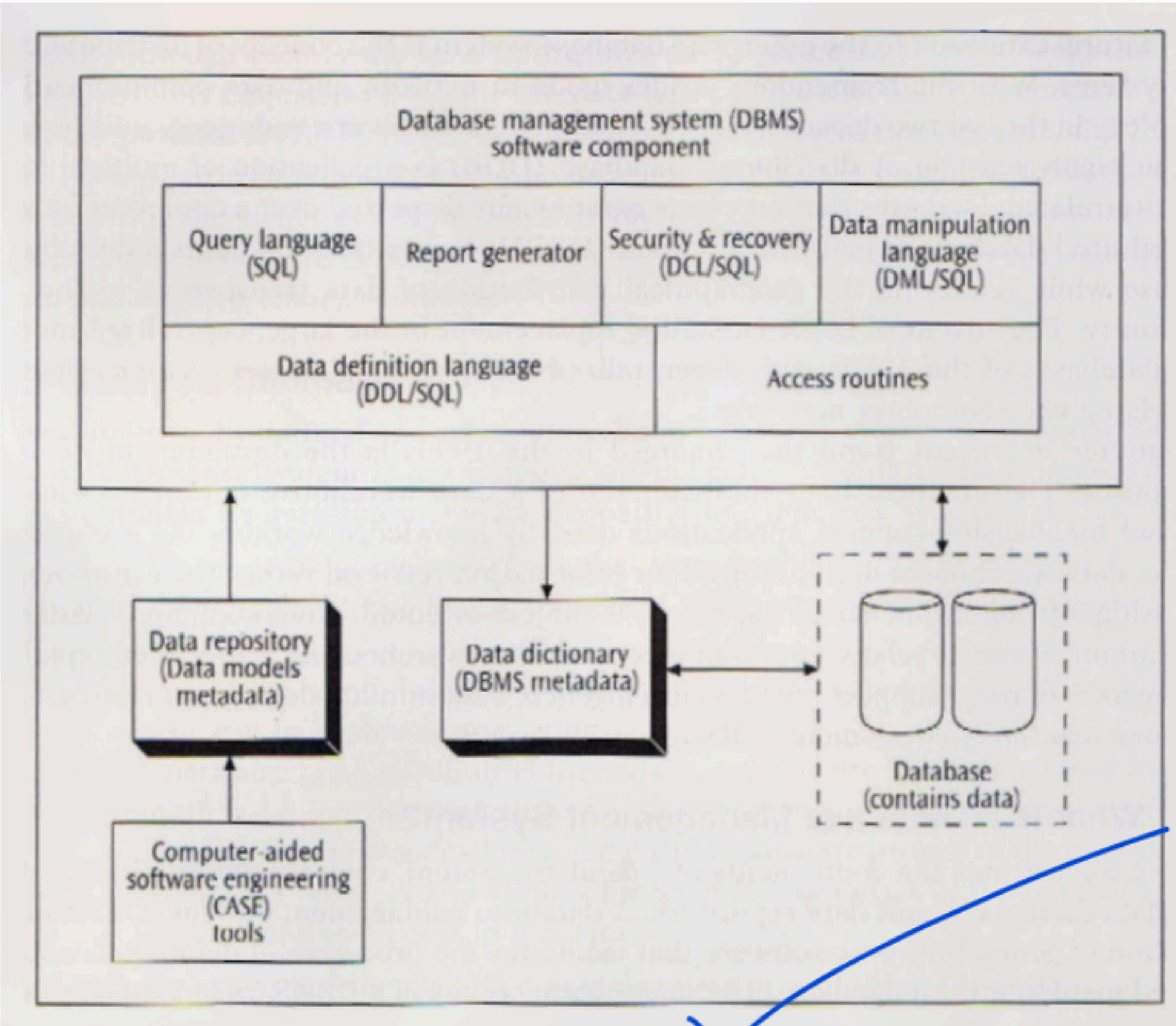


Figure 1.5 Components of a database system

A database system consists of integrated components working together to manage data:

1. **Hardware:** Physical devices like servers, storage disks, and network devices.
2. **Software:**

- **Database Management System (DBMS):** Core software to define, create, manipulate, control, and manage the database. Includes:
    - **Query Processor:** Interprets and executes queries (DDL interpreter, DML compiler, query evaluation engine).
    - **Storage Manager:** Manages physical storage (buffer, file, transaction management).
- **Application Programs & Utilities:** Software for data access, manipulation, reporting, backups (e.g., report generators, data import/export tools).

3. **Data:** Stored information, including operational data and metadata (data about data, in a data dictionary).
4. **Users:**
    - **Database Administrators (DBAs):** Manage the overall system (security, backup, recovery, performance).
    - **Database Designers:** Define database structure (schema).
    - **Application Programmers:** Develop applications interacting with the database.
    - **End Users:** Access data for queries, updates, and reports.
5. **Procedures:** Instructions and rules for database design and use (e.g., login, backup/recovery methods).

(Based on: DBMD UNIT - 1 and 2 Notes, p.3, Figure 1.5 and related text; general DBMS knowledge)

---

b) Distinguish between primary key, candidate key and super key. (3 Marks)

Answer:

Feature	Superkey	Candidate Key	Primary Key
Definition	Any attribute(s) uniquely identifying a tuple (row).	A minimal superkey; no attribute can be removed without losing uniqueness.	The candidate key chosen by the designer to uniquely identify tuples.
Minimality	Not necessarily minimal; may contain redundant attributes.	Must be minimal.	Must be minimal (as it's a candidate key).
Uniqueness	Guarantees uniqueness.	Guarantees uniqueness.	Guarantees uniqueness.
Number	Many per relation.	One or more per relation.	Only one per relation.
Null Values	Generally should not be null (PK enforces no nulls).	Attributes should ideally not be null.	Cannot contain null values (Entity Integrity Constraint).
Example	If {SID, CID} is a candidate key, then	In Employee table, {EmpID} and {SSN}	From {EmpID}, {SSN}, DBA might choose

Feature	Superkey	Candidate Key	Primary Key
	{SID, CID, SName} is a superkey.	could be candidate keys.	{EmpID} as primary key.

(Based on: DBMD UNIT - 1 and 2 Notes, p.8 "Unique Identifiers (Keys)", p.42 "Key Constraints")

---

c) Explain the advanced data manipulation using SQL. (3 Marks)

Answer:
Advanced Data Manipulation Language (DML) in SQL extends beyond basic `INSERT`, `SELECT`, `UPDATE`, `DELETE` for single rows, involving:

1. **Complex Queries:**
    - **Joins:** Combining data from multiple tables (e.g., `INNER JOIN`, `LEFT JOIN`).
    - **Subqueries (Nested Queries):** Queries within other SQL queries for complex filtering or calculations, including correlated subqueries.
    - **Aggregate Functions with Grouping:** Using `COUNT()`, `SUM()`, etc., with `GROUP BY` for group calculations, and `HAVING` to filter groups.
2. **Set Operations:** Combining `SELECT` results using `UNION`, `UNION ALL`, `INTERSECT`, `EXCEPT` (or `MINUS`).
3. **Window Functions (Advanced):** Calculations across a set of table rows related to the current row (e.g., ranking, moving averages) without collapsing rows like `GROUP BY`.
4. **Common Table Expressions (CTEs):** Using `WITH` to define temporary, named result sets for readability and modularity in complex queries.
5. **Bulk Operations:** Efficiently inserting, updating, or deleting large data volumes (e.g., `INSERT INTO ... SELECT ...`).

These features enable sophisticated data retrieval, analysis, and modification for reporting, business intelligence, and complex applications.

(Based on: DBMD UNIT - 3 and 4 Notes, p.12, p.15; general SQL knowledge extending beyond basic DML definitions in Unit 1 notes)

---

d) Explain the Data Control Language (DCL) commands. (3 Marks)

Answer:
Data Control Language (DCL) commands in SQL manage permissions and control access to database objects. Main DCL commands:

1. **GRANT:**

- **Purpose:** Gives specific privileges (e.g., `SELECT`, `INSERT`) on database objects (tables, views) to users or roles.
- **Syntax (Simplified):** `GRANT privilege_list ON object_name TO user_list [WITH GRANT OPTION];`
- **`WITH GRANT OPTION`**: Allows grantee to grant received privileges to others.
- **Example:** `GRANT SELECT, INSERT ON Employees TO 'john_doe';`

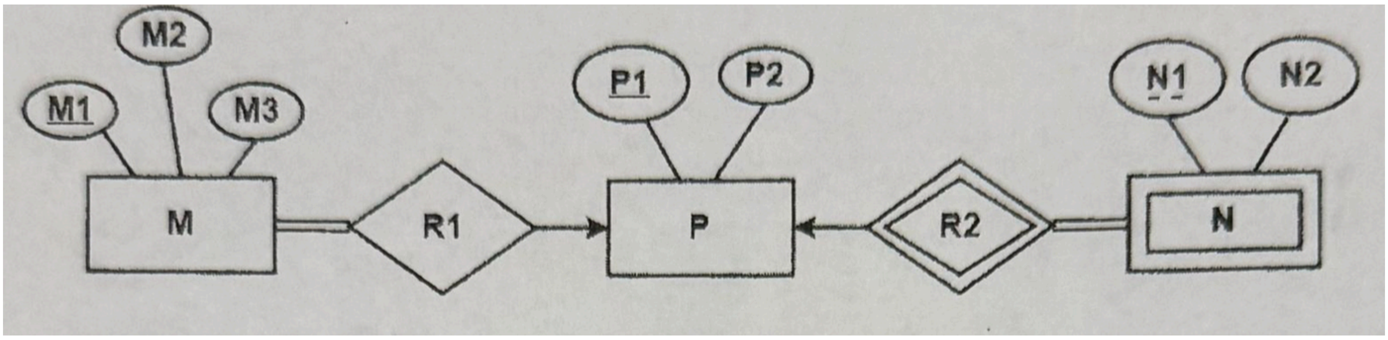
2. **REVOKE:**
    - **Purpose:** Removes previously granted or denied privileges from users or roles.
    - **Syntax (Simplified):** `REVOKE [GRANT OPTION FOR] privilege_list ON object_name FROM user_list [CASCADE | RESTRICT];`
    - **`GRANT OPTION FOR`**: Revokes only the grant ability, not the privilege itself.
    - **`CASCADE`**: Revokes privilege from user and from others to whom this user granted it.
    - **Example:** `REVOKE INSERT ON Employees FROM 'john_doe';`

DCL commands are vital for database security, ensuring users perform only authorized actions.

(Based on: DBMD UNIT - 1 and 2 Notes, p.4; DBMD UNIT - 3 and 4 Notes, p.32-34)

---

e) Find the minimum number of tables required to represent the given ER diagram in the relational model. (3 Marks)



Answer:

Mapping ER diagram to tables:

1. **Entity M (Strong):** Attributes M1 (PK), M2, M3.
    - Rule: Strong entity -> own table.
    - **Table 1: Table_M (M1, M2, M3)**
2. **Entity P (Strong):** Attributes P1 (PK), P2.
    - Rule: Strong entity -> own table.
    - **Table 2: Table_P (P1, P2)**
3. **Entity N (Weak):** Attributes N1 (Partial Key), N2. Identified by P via R2.
    - Rule: Weak entity -> own table. PK = PK of owner (P) + partial key (N1).

- **Table 3: Table_N (P1_FK, N1, N2)** (P1_FK references P)

4. **Relationship R1 (M to P, 1:N):**
   - Rule (1:N): PK of "1" side (M1 from M) becomes FK in "N" side table (P).
   - Table_P updated: **Table_P (P1, P2, M1_FK)** (M1_FK references M)
   - No new table for 1:N.
5. **Relationship R2 (P to N, M:1, Identifying):**
   - Rule (Identifying): Already handled by including P's PK (P1_FK) in Table_N's composite PK.
   - No additional table needed.

Consolidated Tables:

1. **Table_M (M1, M2, M3)**
2. **Table_P (P1, P2, M1_FK)** (M1_FK references Table_M)
3. **Table_N (P1_FK, N1, N2)** (P1_FK references Table_P; {P1_FK, N1} is PK)

Minimum number of tables required is **3**.

(Based on: ER to Relational Mapping rules covered in DBMD UNIT - 1 and 2 Notes, p.43-44)

UNIT-I
(Select one question from Q2 or Q3)

Question 2:

a) Explain the database systems architecture with a suitable diagram. (8 Marks)

Answer:

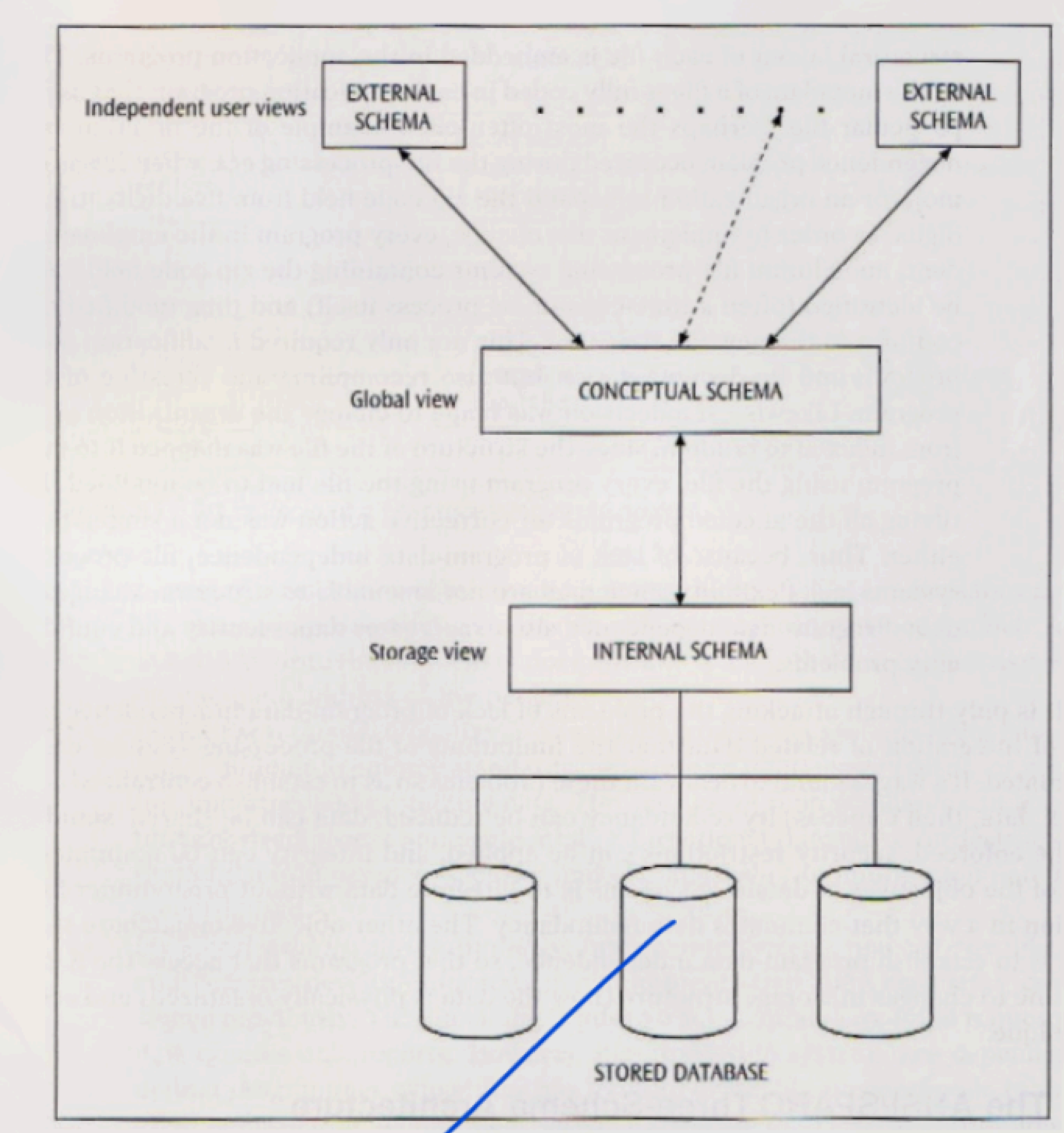


Figure 1.2 The ANSI/SPARC three-schema architecture

The ANSI/SPARC three-schema architecture promotes data independence by separating user views from physical storage. It has three levels:

1. **External Level (User Views / Subschemas):**
   - Closest to users; describes the database part relevant to a specific user/group.
   - Different users can have different views (e.g., HR sees salaries, project managers see skills).
   - Defined by an external schema; provides security by hiding irrelevant/sensitive data.
2. **Conceptual Level (Global View / Conceptual Schema):**

    - Community view of the entire database; describes its logical structure for all users.
    - Defines entities, attributes, relationships, constraints, and semantic information.
    - Technology-independent; hides physical storage details. Database designers work at this level.
3. **Internal Level (Physical View / Internal Schema):**
    - Lowest level, closest to physical storage; describes how data is physically stored.
    - Specifies data structures, file organizations (e.g., B+-trees), access paths (indexes), compression, encryption.
    - Technology-dependent; deals with storage efficiency and access.

Data Independence:

- **Logical Data Independence:** Modify conceptual schema without changing external schemas/applications (e.g., add an attribute).
- **Physical Data Independence:** Modify internal schema without changing conceptual/external schemas (e.g., change file organization, add index).

(Based on: DBMD UNIT - 1 and 2 Notes, p.2, Figure 1.2 and related text)

---

b) Describe the database design life cycle. (7 Marks)

Answer:

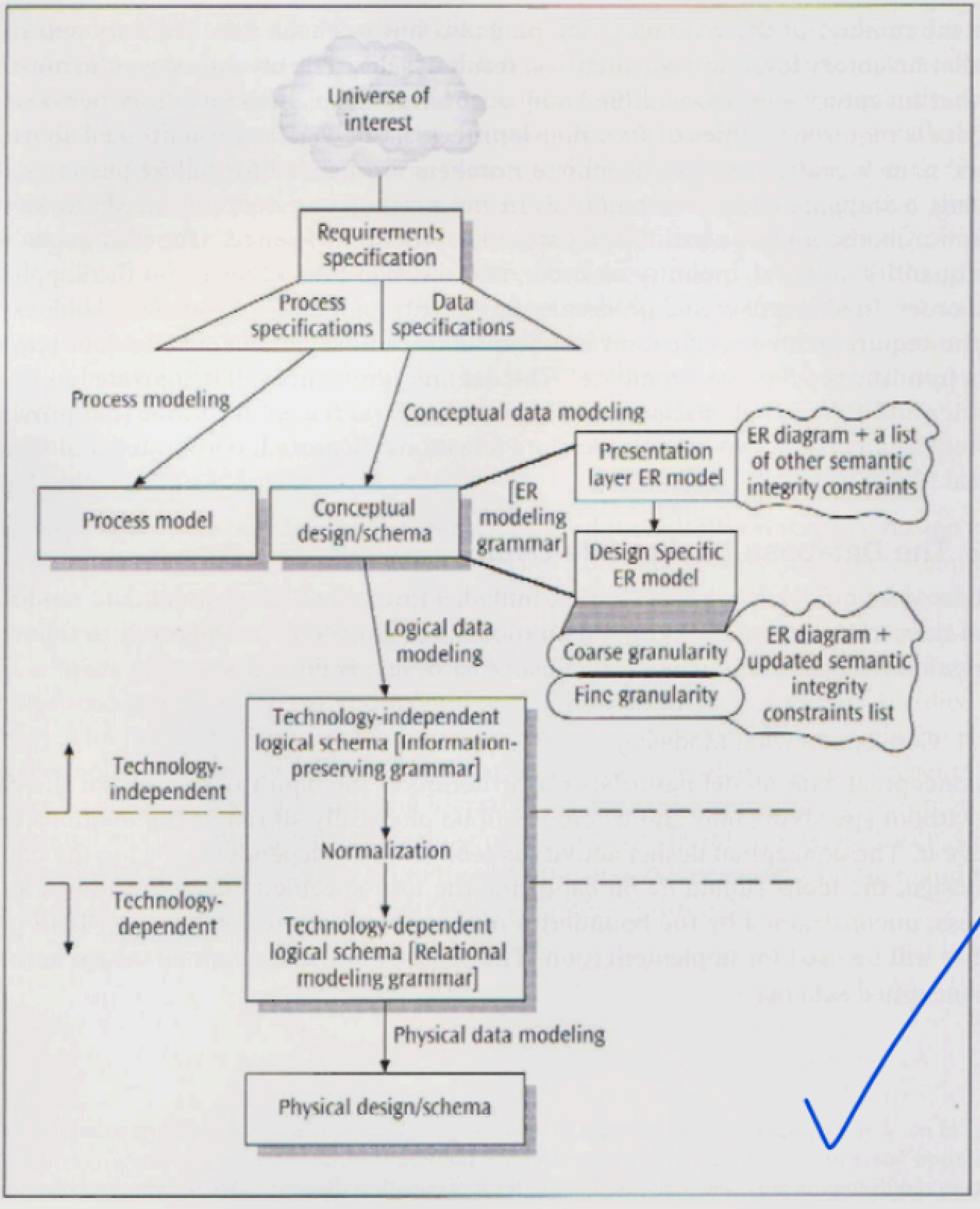


Figure 1.7 Data modeling/database design life cycle

The database design life cycle (DDLC) is a systematic process for designing, implementing, and maintaining a database:

1. **Requirements Specification (and Planning):**
    - **Objective:** Understand and document user data requirements and application needs.
    - **Activities:** Interview users, review documents, identify objectives, entities, processes, business rules. Define scope.

- **Output:** Detailed requirements specification document.

2. **Conceptual Data Modeling (Technology-Independent):**
   - **Objective:** Create a high-level data structure description (entities, attributes, relationships) without physical storage or DBMS specifics.
   - **Activities:** Translate requirements to a conceptual model (e.g., ERD, EERD). Identify types, attributes, constraints.
   - **Output:** Conceptual schema (e.g., ER/EER diagram), validated with users.
3. **Logical Data Modeling (Technology-Dependent):**
   - **Objective:** Transform conceptual schema into a logical model for a chosen data model (typically relational).
   - **Activities:** Map ER/EER to tables, attributes to columns, define PKs/FKs. Apply normalization (1NF, 2NF, 3NF, BCNF).
   - **Output:** Logical schema (set of normalized relational tables).
4. **Physical Data Modeling (DBMS-Specific):**
   - **Objective:** Specify internal storage structures, access paths, file organizations for the chosen DBMS.
   - **Activities:** Define column data types, design indexes, consider file organization, clustering, partitioning. Estimate storage, plan security/recovery.
   - **Output:** Physical schema (internal schema, DDL statements).
5. **Implementation and Testing:**
   - **Objective:** Create and test the database.
   - **Activities:** Write DDL, populate data, develop/test applications, perform unit, integration, performance tests.
6. **Deployment, Operation, and Maintenance:**
   - **Objective:** Deploy for operational use and maintain over time.
   - **Activities:** Deploy DB/apps, monitor performance, tune, regular backups, manage security/access, make modifications.

(Based on: DBMD UNIT - 1 and 2 Notes, p.5, Figure 1.7 and related text)

Question 3:

a) Explain the data models available for the database modelling system. (8 Marks)

Answer:
Data models are tools to describe database structure, data types, relationships, and constraints:

1. **Hierarchical Data Model:**

- **Structure:** Tree-like; parent-child relationships (1 parent per child).
- **Pros:** Simple for some data, efficient hierarchical access.
- **Cons:** Inflexible (M:N complex), restricted data access.
- **Relevance:** Historical (e.g., IBM's IMS).

2. **Network Data Model:**
   - **Structure:** Graph-like; records can have multiple parents/children.
   - **Pros:** More flexible than hierarchical, represents M:N better.
   - **Cons:** Complex design/management, navigation via pointers.
   - **Relevance:** Historical (e.g., IDMS).
3. **Relational Data Model:**
   - **Structure:** Data in tables (relations) with rows (tuples) and columns (attributes). Relationships via common columns (PKs/FKs).
   - **Pros:** Simple, high data independence, flexible querying (SQL), strong foundation.
   - **Cons:** Can be slower for complex joins if unoptimized.
   - **Relevance:** Most widely used (e.g., Oracle, MySQL, SQL Server).
4. **Entity-Relationship (ER) Data Model (Conceptual):**
   - **Structure:** High-level; real world as entities (objects) and relationships.
   - **Representation:** Entities (rectangles), attributes (ovals), relationships (diamonds).
   - **Pros:** Easy to understand/communicate design.
   - **Cons:** Not directly implemented; a design tool mapped to logical models.
   - **Relevance:** Standard for conceptual design.
5. **Enhanced Entity-Relationship (EER) Data Model (Conceptual):**
   - **Structure:** Extends ER with superclasses/subclasses, specialization/generalization, inheritance, categorization.
   - **Pros:** More precise modeling for complex data.
   - **Relevance:** Designing complex databases with hierarchies.
6. **Object-Oriented Data Model:**
   - **Structure:** Data as objects (instances of classes) encapsulating data and behavior (methods). Supports inheritance, polymorphism.
   - **Pros:** Represents complex types/relationships from OOP.
   - **Cons:** Less mature than relational, complex querying.
   - **Relevance:** OODBMS, ORDBMS (e.g., CAD/CAM).
7. **Object-Relational Data Model:**

    - **Structure:** Hybrid; combines relational features with object-oriented concepts (user-defined types, complex objects).
    - **Pros:** Extends relational power, SQL compatibility.
    - **Relevance:** Many modern RDBMSs (e.g., PostgreSQL, Oracle).
8. **NoSQL Data Models (e.g., Document, Key-Value, Column-family, Graph):**
    - **Structure:** Non-relational; for scalability, flexibility, specific workloads.
        - **Document:** Data in documents (e.g., JSON).
        - **Key-Value:** (key, value) pairs.
        - **Column-family:** Data in columns (good for sparse data).
        - **Graph:** Nodes, edges, properties (for connected data).
    - **Pros:** Scalable, flexible schema, good for unstructured/semi-structured data.
    - **Cons:** Often eventual consistency, varied query capabilities.
    - **Relevance:** Growing for big data, real-time apps.

(Based on: General knowledge of data models; ER and EER are covered in DBMD UNIT - 1 and 2 Notes, p.6, p.27. Relational model is the basis for much of the notes.)

---

b) Illustrate the design issues in ER & EER modelling. (7 Marks)

Answer:
Effective ER/EER modeling requires addressing several design issues:

1. **Choosing Entity vs. Attribute:**
    - **Issue:** Model a concept as an entity or an attribute.
    - **Guideline:** Entity if it has own attributes or independent relationships; attribute if a simple property.
    - **Example:** "Address" for `Employee`. Attribute if simple; entity if multiple/complex addresses shared.
2. **Choosing Entity vs. Relationship:**
    - **Issue:** Represent a concept as an entity or a relationship (esp. M:N with attributes).
    - **Guideline:** If an association has descriptive attributes, model as an (associative) entity.
    - **Example:** `WORKS_ON` (Employee-Project) with `HoursWorked`. Can be M:N relationship with attribute, or `Assignment` associative entity.
3. **Binary vs. Higher-Degree Relationships (Ternary, N-ary):**
    - **Issue:** Degree of a relationship.
    - **Guideline:** Use higher-degree sparingly, only if truly irreducible. Often decomposable into binaries.

    - **Example:** `SUPPLIES` (Supplier, Part, Project). Ternary if supplier supplies a part *only* for a specific project.
    - *(See DBMD UNIT - 1 and 2 Notes, p.33-34).*
4. **Use of Weak Entities:**
    - **Issue:** When to use a weak entity (existence-dependent, identified via owner).
    - **Guideline:** Entity's existence depends on another, PK includes owner's PK.
    - **Example:** `Dependents` of an `Employee`. Identified by `(EmployeeID, DependentName)`.
    - *(See DBMD UNIT - 1 and 2 Notes, p.18).*
5. **When to Use EER Constructs (Specialization/Generalization, Categorization):**
    - **Issue:** Use superclass/subclass or categories.
    - **Specialization/Generalization (Is-A):** Entity has distinct subgroups (subclasses) with common/specific attributes/relationships. E.g., `Employee` -> `SalariedEmployee`, `HourlyEmployee`.
    - **Categorization (Union Type):** Subclass is subset of *union* of distinct entity types. E.g., `VehicleOwner` is `Person` OR `Company`.
    - *(See DBMD UNIT - 1 and 2 Notes, p.27, p.29, p.32-33).*
    - **Constraints for Specialization:** Disjointness (d/o), completeness (total/partial).
6. **Handling M:N Relationships and Multi-valued Attributes (Design-Specific ER):**
    - **Issue:** Mapping M:N and multi-valued attributes to relational models.
    - **Guideline:** M:N -> associative entity + two 1:N. Multi-valued attribute -> separate entity + 1:N.
    - *(See DBMD UNIT - 1 and 2 Notes, p.24, p.26).*
7. **Validation of Conceptual Design (Connection Traps):**
    - **Issue:** ER structure leading to misinterpretation.
    - **Fan Trap:** Ambiguous pathway between entity occurrences due to multiple 1:N relationships fanning out. E.g., `Division` has many `Staff` and many `Branches` (which staff at which branch?).
    - **Chasm Trap:** Suggested relationship pathway doesn't exist for all occurrences due to optional participation. E.g., `Branch` -> `Staff` (optional manages) -> `Property`.
    - *(See DBMD UNIT - 1 and 2 Notes, p.37-40).*

Thoughtful handling of these issues leads to well-structured, accurate database models.

UNIT-II
(Select one question from Q4 or Q5)

Question 4:

a) Explain the ER model and EER Model to map with logical schema. (8 Marks)

Answer:
Mapping ER/EER models to a logical (typically relational) schema converts conceptual constructs into tables, columns, and keys.

Mapping ER Model Constructs:

1. **Strong Entity Types:** Create a table; entity attributes become columns; choose a PK. Composite attributes map to simple component columns.
2. **Weak Entity Types:** Create a table; include attributes. PK = PK of owner entity (as FK) + partial key of weak entity.
3. **1:1 Relationship Types:**
    - **Foreign Key:** Add PK of one entity as a unique FK in the other (prefer side with total participation).
    - **Merged Relation:** If both total participation, consider merging into one table (less common).
    - **Relationship Relation:** Separate table for relationship (PK of one entity as PK, PK of other as unique FK); relationship attributes here.
4. **1:N Relationship Types:** In N-side table, include PK of 1-side entity as FK. Relationship attributes also go in N-side table.
5. **M:N Relationship Types:** Create a new junction/associative table. PK = composite of PKs from both participating entities (as FKs). Relationship attributes go in this new table.
6. **Multi-valued Attributes:** Create a new table. Include PK of original entity (as FK) and the multi-valued attribute. PK of new table = (FK, multi-valued attribute).

Mapping EER Model Constructs:

1. **Specialization/Generalization (Superclass/Subclass - SC/sc):**
    - **Option 1 (Multiple Relations - SC and sc's):** Table for SC, table for each sc. SC's PK is PK for each sc (also FK to SC). Good for disjoint subclasses with specific attributes.
    - **Option 2 (Single Relation - SC only):** One table for SC. Include all attributes of SC and all sc's. Use nulls and a "type" attribute. Good for overlapping or few specific sc attributes.
    - **Option 3 (Multiple Relations - sc's only):** Table for each sc, including inherited SC attributes. Only if SC participation is total and disjoint (can cause redundancy).
2. **Specialization Hierarchy/Lattice:** Apply chosen SC/sc mapping recursively. Shared subclass (lattice) might have multiple FKs (if Option 1 used).
3. **Categorization (Union Type):** Create table for category (subclass) with surrogate PK. This PK becomes FK in superclass tables. Complex, often avoided.

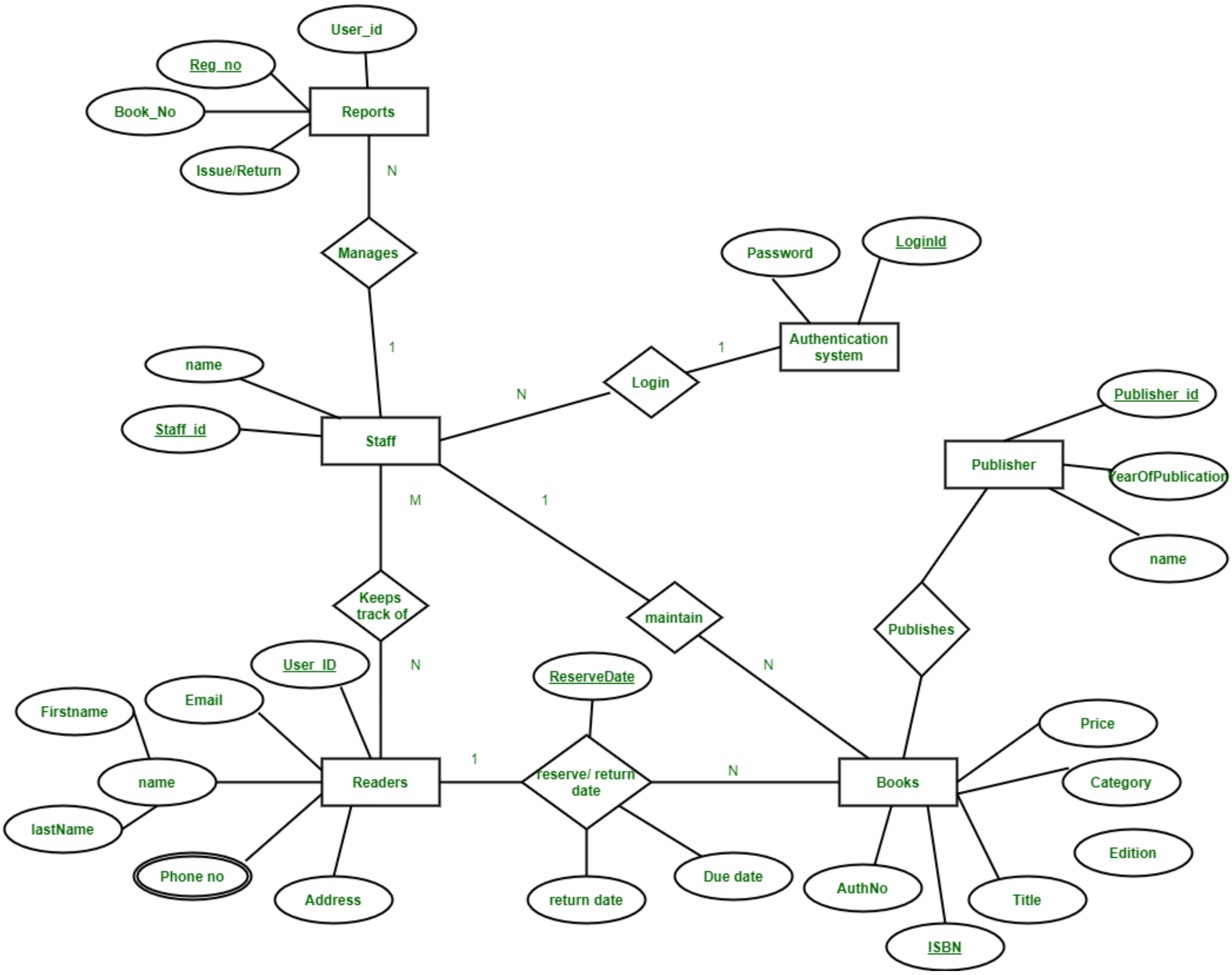
(Based on: DBMD UNIT - 1 and 2 Notes, p.43-44 for ER, p.45, p.47-49 for EER)

---

b) Construct an ER diagram for the Library Management System covering all major activities and map its logical schema. (7 Marks)

Answer:

ER Diagram for Library Management System:



Interpretation for Logical Schema Mapping:

- PKs: `User_id` (Reports), `Staff_id`, `LoginId`, `Publisher_id`, `User_ID` (Readers), `ISBN` (Books) are PKs.
- "Reports" entity: Represents transactions (issue/return). `Reg_no` likely transaction ID.
- "reserve/return date" M:N relationship (Readers-Books) with attributes -> separate table.
- "maintain" relationship: Staff involved in loan process.
- "Phone no" (Readers): Multi-valued attribute -> separate table.
- "name" (Staff): Split into `StaffFirstName`, `StaffLastName`.
- `AuthNo` (Books): Simple attribute (ideally FK to Authors table not shown).
- `YearOfPublication`: Attribute of `Publisher` (e.g., founding year).

Logical Schema (Relational Tables):

1. **AuthenticationSystem_Table**
   - `LoginID` (PK, VARCHAR(50))
   - `Password` (VARCHAR(255), Hashed)
2. **Staff_Table**
   - `StaffID` (PK, VARCHAR(20))
   - `StaffFirstName` (VARCHAR(50), NOT NULL)
   - `StaffLastName` (VARCHAR(50), NOT NULL)
   - `LoginID` (FK, VARCHAR(50), References AuthenticationSystem_Table(LoginID))
3. **Publisher_Table**
   - `PublisherID` (PK, VARCHAR(20))
   - `PublisherName` (VARCHAR(100), NOT NULL, UNIQUE)
   - `YearOfPublication` (INT)
4. **Books_Table**
   - `ISBN` (PK, VARCHAR(13))
   - `Title` (VARCHAR(255), NOT NULL)
   - `AuthNo` (VARCHAR(50))
   - `Edition` (VARCHAR(50))
   - `Category` (VARCHAR(50))
   - `Price` (DECIMAL(10,2))
   - `PublisherID` (FK, VARCHAR(20), References Publisher_Table(PublisherID))
5. **Readers_Table**
   - `UserID` (PK, VARCHAR(20))
   - `FirstName` (VARCHAR(50), NOT NULL)
   - `LastName` (VARCHAR(50), NOT NULL)
   - `Email` (VARCHAR(100), UNIQUE)
   - `Address` (VARCHAR(255))
6. **Reader_Phones_Table** (For multi-valued Phone no)
   - `UserID` (FK, PK_part, VARCHAR(20), References Readers_Table(UserID))
   - `PhoneNumber` (PK_part, VARCHAR(20))
   - *Primary Key: (`UserID`, `PhoneNumber`)*
7. **Book_Loan_Transaction_Table** (From "Reports", "reserve/return date", "Manages", "maintain")

- `TransactionID` (PK, INT or VARCHAR(30)) /* e.g., `Reg_no` */
- `Reader_UserID` (FK, VARCHAR(20), References Readers_Table(UserID), NOT NULL)
- `Book_ISBN` (FK, VARCHAR(13), References Books_Table(ISBN), NOT NULL)
- `Issuing_StaffID` (FK, VARCHAR(20), References Staff_Table(StaffID), NOT NULL)
- `ReserveDate` (DATE, Nullable)
- `IssueDate` (DATE)
- `DueDate` (DATE, NOT NULL if issued)
- `ReturnDate` (DATE, Nullable)
- `Status` (VARCHAR(20), e.g., 'Issued', 'Returned') /* `Issue/Return` attribute */

8. **Staff_Reader_Monitoring_Table** (From "Keeps track of" M:N relationship)
    - `StaffID` (FK, PK_part, VARCHAR(20), References Staff_Table(StaffID))
    - `Reader_UserID` (FK, PK_part, VARCHAR(20), References Readers_Table(UserID))
    - `MonitoringStartDate` (DATE, Optional)
    - *Primary Key: (`StaffID`, `Reader_UserID`)*

(Based on ER design principles and mapping rules from DBMD UNIT - 1 and 2 Notes)

Question 5:

a) Define Normalization. Explain the types of Normalization. (8 Marks)

Answer:
Definition of Normalization:
Normalization is organizing database data to reduce redundancy and improve data integrity. It involves decomposing tables into smaller, well-structured ones with defined relationships.
Goals: Minimize redundancy, eliminate Insert/Update/Delete anomalies, ensure logical data dependencies, produce flexible/maintainable design.

Types of Normalization (Normal Forms):

1. **First Normal Form (1NF):**
    - **Rule:** All attribute values are atomic (single, indivisible). No repeating groups or multi-valued attributes in a cell.
    - *(See DBMD UNIT - 1 and 2 Notes, p.51-52).*
2. **Second Normal Form (2NF):**
    - **Rule:** In 1NF, and every non-primary-key attribute is fully functionally dependent on the *entire* primary key (no partial dependencies).
    - *(See DBMD UNIT - 1 and 2 Notes, p.52-53).*

3. **Third Normal Form (3NF):**
    - **Rule:** In 2NF, and no transitive dependencies (non-key attribute depends on another non-key attribute).
    - *(See DBMD UNIT - 1 and 2 Notes, p.53-54).*
4. **Boyce-Codd Normal Form (BCNF):**
    - **Rule:** In 3NF, and for every non-trivial functional dependency `X -> Y`, `X` must be a superkey. Stricter than 3NF.
    - *(See DBMD UNIT - 1 and 2 Notes, p.54-55).*
5. **Fourth Normal Form (4NF):**
    - **Rule:** In BCNF, and no non-trivial multi-valued dependencies (MVDs `X ->> Y`), unless `X` is a superkey. Separates independent multi-valued facts.
    - *(See DBMD UNIT - 1 and 2 Notes, p.55-57).*
6. **Fifth Normal Form (5NF / Project-Join Normal Form - PJ/NF):**
    - **Rule:** In 4NF, and no join dependencies (JDs) not implied by candidate keys. Ensures no further lossless decomposition possible (except by candidate keys).
    - *(See DBMD UNIT - 1 and 2 Notes, p.57-58).*

3NF or BCNF is often sufficient for most designs.

---

b) Explain the Mapping of higher degree relationships. (7 Marks)

Answer:
Higher-degree relationships (ternary, n-ary) involve three or more entity types in a single relationship instance.

General Approach: Associative Entity (Junction Table)
Map any higher-degree relationship by creating a new relation (associative entity/junction table) for the relationship.

Steps for Mapping an N-ary Relationship:

1. **Create New Relation:** For n-ary relationship `R`, create table `S`.
2. **Include Foreign Keys:** For each participating entity `E1, ..., En`, include `PK(Ei)` as an FK in `S`.
3. **Primary Key of `S`:** Typically, the combination of all these FKs: `{PK(E1), ..., PK(En)}`.
4. **Map Relationship Attributes:** Attributes of `R` become columns in `S`.
5. **Cardinality Constraints:** Exact (min,max) constraints for n-ary relationships are hard to enforce with only PK/FK; may need application logic or complex checks.

Example: Ternary Relationship `SCHEDULE` (Instructor, Course, Quarter)
Entities: `INSTRUCTOR(InstructorID_PK, ...)`, `COURSE(CourseID_PK, ...)`, `QUARTER(QuarterID_PK, ...)`
Relationship `SCHEDULE` has attribute `RoomNo`.

Mapping `SCHEDULE`:

```
CREATE TABLE SCHEDULE_TABLE (
    InstructorID_FK INT,        -- FK to INSTRUCTOR
    CourseID_FK VARCHAR(10),    -- FK to COURSE
    QuarterID_FK VARCHAR(10),   -- FK to QUARTER
    RoomNo VARCHAR(10),         -- Attribute of SCHEDULE
    PRIMARY KEY (InstructorID_FK, CourseID_FK, QuarterID_FK),
    FOREIGN KEY (InstructorID_FK) REFERENCES INSTRUCTOR(InstructorID_PK),
    FOREIGN KEY (CourseID_FK) REFERENCES COURSE(CourseID_PK),
    FOREIGN KEY (QuarterID_FK) REFERENCES QUARTER(QuarterID_PK)
);
```

Considerations:

- **Decomposition:** Evaluate if the higher-degree relationship can be losslessly decomposed into binary ones. Preferable if semantics are preserved.
- **Conceptual Model First:** (DBMD Notes, p.50) Suggests decomposing into a gerund (associative) entity in the conceptual model first, then mapping this entity and its binary relationships.
- **Aggregation:** Distinct concept where a relationship is treated as a higher-level entity.

The key is to accurately capture all instances and attributes of the higher-degree relationship.

UNIT-III
(Select one question from Q6 or Q7)

Question 6:

a) Describe the database creation using SQL with help of suitable examples. (8 Marks)

Answer:
Database creation in SQL mainly uses the `CREATE TABLE` DDL statement to define a new table, its columns, data types, and constraints.

Syntax of `CREATE TABLE` (Simplified):

```
CREATE TABLE table_name (
    column_name_1 data_type [column_constraints],
    column_name_2 data_type [column_constraints],
```

```
    ...
    [table_constraints]
);
```

Key Components:

- `table_name`: Name of the new table.
- `column_name`: Name of a column.
- `data_type`: Type of data (e.g., `INTEGER`, `VARCHAR(n)`, `DATE`, `DECIMAL(p,s)`).
- `column_constraints`: Apply to individual columns (e.g., `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `CHECK (condition)`, `DEFAULT default_value`, `REFERENCES referenced_table`).
- `table_constraints`: Apply to one or more columns (e.g., `PRIMARY KEY (cols)`, `UNIQUE (cols)`, `FOREIGN KEY (cols) REFERENCES ...`, `CHECK (condition)`).

Example: Creating Tables for a Medical System
(Based on Figure 10.1a, Box 1, Box 2, Box 3 in notes)

1. Patient Table:

```sql
CREATE TABLE Patient (
    Pat_p_alpha CHAR(2) NOT NULL,
    Pat_p_num CHAR(5) NOT NULL,
    Pat_name VARCHAR(100) NOT NULL,
    Pat_gender CHAR(1) CHECK (Pat_gender IN ('M', 'F')),
    Pat_age SMALLINT CHECK (Pat_age >= 0 AND Pat_age <= 120),
    Pat_admit_date DATE,
    Pat_wing CHAR(1),
    Pat_room_num INT,
    Pat_bed CHAR(1) CHECK (Pat_bed IN ('A', 'B')),
    PRIMARY KEY (Pat_p_alpha, Pat_p_num)
);
```

2. Medication Table:

```sql
CREATE TABLE Medication (
    Med_code CHAR(5) PRIMARY KEY,
    Med_name VARCHAR(100) NOT NULL UNIQUE,
    Med_unit_price DECIMAL(5,2) CHECK (Med_unit_price > 0),
    Med_qty_onhand INT DEFAULT 0,
    Med_qty_onorder INT DEFAULT 0,
    CONSTRAINT chk_med_stock CHECK ((Med_qty_onhand + Med_qty_onorder) BETWEEN 0
AND 5000)
);
```

3. Order Table (linking Patient and Medication):

```sql
CREATE TABLE Orders (
    Ord_rx_num CHAR(13) PRIMARY KEY,
    Ord_pat_p_alpha CHAR(2) NOT NULL,
    Ord_pat_p_num CHAR(5) NOT NULL,
    Ord_med_code CHAR(5) NOT NULL,
    Ord_dosage SMALLINT DEFAULT 1 CHECK (Ord_dosage IN (1, 2, 3)),
    Ord_freq SMALLINT DEFAULT 1 CHECK (Ord_freq IN (1, 2, 3)),
    FOREIGN KEY (Ord_pat_p_alpha, Ord_pat_p_num) REFERENCES Patient(Pat_p_alpha,
Pat_p_num)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (Ord_med_code) REFERENCES Medication(Med_code)
        ON DELETE RESTRICT ON UPDATE RESTRICT
);
```

This shows tables with various data types, PKs (simple/composite), FKs with referential actions, CHECK, and DEFAULT constraints.

(Based on: DBMD UNIT - 3 and 4 Notes, p.1-8)

---

b) Explain briefly DDL and DML with syntax and suitable examples. (7 Marks)

Answer:

DDL (Data Definition Language):
Defines, modifies, and removes database structures (schemas) and objects.

- **Key DDL Commands:**
  1. `CREATE`: Creates new DB objects (tables, views).
     - **Syntax (Table):** `CREATE TABLE table_name (col1 datatype, ...);`
     - **Example:** `CREATE TABLE Products (ProdID INT PRIMARY KEY, Name VARCHAR(50));`
  2. `ALTER`: Modifies existing DB objects.
     - **Syntax (Add Column):** `ALTER TABLE table_name ADD new_column datatype;`
     - **Example:** `ALTER TABLE Products ADD Price DECIMAL(10,2);`
  3. `DROP`: Deletes existing DB objects.
     - **Syntax (Table):** `DROP TABLE table_name;`
     - **Example:** `DROP TABLE Products;`
  4. `TRUNCATE`: Removes all rows from a table (structure remains); faster than `DELETE` for all rows.
     - **Syntax (Table):** `TRUNCATE TABLE table_name;`
     - **Example:** `TRUNCATE TABLE TempData;`

DML (Data Manipulation Language):
Retrieves, inserts, updates, and deletes data within tables.

- **Key DML Commands:**
  1. `SELECT`: Retrieves data.
     - **Syntax:** `SELECT col_list FROM table_name [WHERE condition];`
     - **Example:** `SELECT Name, Price FROM Products WHERE Price > 100;`
  2. `INSERT`: Adds new rows.
     - **Syntax:** `INSERT INTO table_name (col1, col2) VALUES (val1, val2);`
     - **Example:** `INSERT INTO Products (ProdID, Name, Price) VALUES (1, 'Laptop', 1200);`
  3. `UPDATE`: Modifies existing data.
     - **Syntax:** `UPDATE table_name SET col1 = val1 [WHERE condition];`
     - **Example:** `UPDATE Products SET Price = 1150 WHERE ProdID = 1;`
  4. `DELETE`: Removes rows.
     - **Syntax:** `DELETE FROM table_name [WHERE condition];`
     - **Example:** `DELETE FROM Products WHERE ProdID = 1;`

(Based on: DBMD UNIT - 1 and 2 Notes, p.4; DBMD UNIT - 3 and 4 Notes, p.1, p.8, p.12)

Question 7:

a) Describe the cursor and type of cursor. What is the need of cursor in database programming? (8 Marks)

Answer:
Cursor:
A cursor is a database control structure enabling row-by-row traversal over a result set returned by an SQL query. It acts as a pointer to a specific row, facilitating procedural processing of set-based SQL results.

Need for Cursors: (DBMD UNIT - 3 and 4 Notes, p.18)
SQL is set-oriented, while host languages (C, Java) are record-oriented, creating an **impedance mismatch**. Cursors bridge this by allowing applications to:

1. Define a row set (from an SQL query).
2. Iterate through this set, fetching one row at a time into host variables.
3. Potentially update/delete the current row (with updatable cursors).

Operations on Cursors: (DBMD UNIT - 3 and 4 Notes, p.18)
`DECLARE` (defines cursor with query), `OPEN` (executes query, positions cursor), `FETCH` (retrieves row,

advances cursor), `UPDATE ... WHERE CURRENT OF` (modifies current row), `DELETE ... WHERE CURRENT OF` (deletes current row), `CLOSE` (releases resources).

Types of Cursors: (DBMD UNIT - 3 and 4 Notes, p.19-20)

1. **Read-Only Cursor:** Allows only fetching data. (Default or `FOR READ ONLY`).
2. **Updatable Cursor:** Allows fetching and modifying/deleting current row (`FOR UPDATE [OF column_list]`). Query usually simple.
3. **Forward-Only Cursor (Non-Scrollable):** Default; rows fetched sequentially (first to last). Most efficient.
4. **Scrollable Cursor:** Flexible movement (`NEXT`, `PRIOR`, `FIRST`, `LAST`, `ABSOLUTE n`, `RELATIVE n`). (`SCROLL CURSOR`).
5. **Insensitive Cursor (Snapshot Cursor):** Operates on a temporary copy of data at open time. Changes by others not visible. (`INSENSITIVE CURSOR`). Provides static view.
6. **Sensitive Cursor:** Attempts to reflect changes by others after open. Behavior varies, can be complex.
7. **Keyset-Driven Cursor:** A type of sensitive cursor. Keys of qualifying rows fixed at open. Non-key value changes visible; deleted rows appear as "holes"; new qualifying rows usually not seen.

(Note: Static/Dynamic cursors map broadly to Insensitive/highly Sensitive cursors.)

---

b) What is database trigger? Explain the types of Trigger. (7 Marks)

Answer:
Database Trigger:
A procedural code (SQL block/stored procedure) automatically executed by the DBMS in response to specific DML events (e.g., `INSERT`, `UPDATE`, `DELETE`) or DDL events (`CREATE`) on a table/view.

ECA Model (Event-Condition-Action): (DBMD UNIT - 3 and 4 Notes, p.21)

- **Event:** The DML/DDL operation causing trigger to fire.
- **Condition (Optional):** Boolean expression; action executes if true.
- **Action:** Code executed when event occurs and condition met.

Syntax (Simplified Generic): (DBMD UNIT - 3 and 4 Notes, p.21)

```
CREATE TRIGGER trigger_name {BEFORE | AFTER} {INSERT | DELETE | UPDATE [OF cols]}
ON table_name [FOR EACH ROW] [WHEN (condition)] BEGIN ... END;
```

Types of Triggers:

1. **Row-Level vs. Statement-Level Trigger:**

    - **Row-Level (`FOR EACH ROW`):** Fires once per affected row. Can access `:OLD`/`:NEW` row values. Ex: Auditing each updated row.
    - **Statement-Level (Default):** Fires once per DML statement. Cannot access `:OLD`/`:NEW`. Ex: Alert on any delete attempt from critical table.
2. **Timing (BEFORE vs. AFTER):**
    - **BEFORE Trigger:** Action executes *before* triggering DML. Uses: Validate/modify new data, prevent operations.
    - **AFTER Trigger:** Action executes *after* triggering DML and constraints. Uses: Auditing, complex integrity, propagating changes.
3. **INSTEAD OF Triggers:** (DBMD UNIT - 3 and 4 Notes, p.21)
    - For views (especially non-updatable ones). Fires *instead* of DML on view. Defines how to apply operation to base tables.
4. **DDL Triggers:** (DBMD UNIT - 3 and 4 Notes, p.22)
    - Fire on DDL events (`CREATE TABLE`). Uses: Audit schema changes, enforce naming. (DBMS-specific support).
5. **Logon/Logoff Triggers (System-Level):** (DBMD UNIT - 3 and 4 Notes, p.22)
    - Fire on user connect/disconnect. Uses: Audit sessions, set parameters. (DBMS-specific).
6. **Database Event Triggers (Server-Level):** (DBMD UNIT - 3 and 4 Notes, p.22)
    - Fire on DB events (startup, shutdown, errors). Uses: Admin tasks, custom logging. (DBMS-specific).
7. **Compound Triggers (Oracle specific):** (DBMD UNIT - 3 and 4 Notes, p.22)
    - Defines actions for multiple timing points (`BEFORE STATEMENT`, `BEFORE EACH ROW`, etc.) for one DML event in a single trigger.

Triggers are powerful but can add complexity; use carefully.

UNIT-IV
(Select one question from Q8 or Q9)

Question 8:

a) Explain the Indexing and its methods with the help of suitable examples. (8 Marks)

Answer:
Indexing:
A database technique to speed up row retrieval. An index is a separate data structure (e.g., B+-tree) storing copies of one or more columns (index key) with pointers (rowids) to actual data rows. It helps locate relevant rows quickly without full table scans.

Why Indexing? Avoids inefficient full table scans for large tables by providing a shortcut.

Methods/Types of Indexing:

1. **Primary Index (often Clustered):** Index key specifies data file's sequential order. Table physically ordered by this key. Max one per table.
   - **Example:** Index on `EmployeeID` (PK), employee records physically stored in `EmployeeID` order.
2. **Clustered Index:** Physical order of table rows matches index key order.
   - **Example:** `CREATE CLUSTERED INDEX IX_Orders_Date ON Orders(OrderDate);` `Orders` table rows physically sorted by `OrderDate`. Efficient for range queries on `OrderDate`.
3. **Secondary Index (Non-Clustered Index):** Index key order differs from physical data order. Index entries point to data rows. Multiple per table.
   - **Example:** `CREATE INDEX IX_Emp_LName ON Employees(LastName);` Allows quick lookups by `LastName`.
4. **B+-Tree Index:** Most common; balanced tree. Leaf nodes contain (key, pointer) pairs, linked sequentially. Efficient for equality and range searches.
   - **Example:** Default index type, e.g., `CREATE INDEX IX_Prod_Price ON Products(Price);`
5. **Hash Index:** Uses hash function to find bucket/page for an index entry.
   - **Example:** Index on `CustomerEmail`. Fast for `WHERE CustomerEmail = '...';` Not for range queries.
6. **Unique Index:** Enforces no duplicate values for indexed column(s). PK indexes are unique.
   - **Example:** `CREATE UNIQUE INDEX UQ_Emp_Email ON Employees(Email);`
7. **Composite Index (Multi-column Index):** Index on two or more columns. Column order is key.
   - **Example:** `CREATE INDEX IX_Orders_CustDate ON Orders(CustomerID, OrderDate);` Useful for filters on `CustomerID` or `CustomerID AND OrderDate`.
8. **Covering Index:** Non-clustered index containing all columns for a query (in `SELECT` & `WHERE`). Query answered from index (index-only scan).
   - **Example:** For `SELECT OrderDate, Amount FROM Orders WHERE CustID = 1;`, index on `(CustID, OrderDate, Amount)` is covering.

(Based on: DBMD UNIT - 3 and 4 Notes, p.23-27)

---

b) Discuss about the database security used in the database modelling. (7 Marks)

Answer:
Integrating security into database modeling is crucial for building secure systems.

Security Considerations During Modeling Phases:

1. **Conceptual Level (ER/EER & Requirements):**
    - **Identify Sensitive Data:** Pinpoint entities/attributes needing protection (e.g., PII, financial data).
    - **Define Access Privileges Conceptually (User Roles):** Document roles (HR, Sales) and their intended data access (read, create, modify, delete).
    - **Consider Views for Security:** Note needs for subset views (hiding columns, showing aggregated data) for specific user groups. E.g., view for average salaries, not individual.
    - **Data Ownership:** Clarify who owns data, informing access granting authority.
2. **Logical Level (Relational Schema Design):**
    - **View Design:** Implement views for:
        - **Column-level security:** Hide sensitive columns.
        - **Row-level security:** Filter rows via `WHERE` clause based on user identity/attributes (e.g., manager sees only their region's orders).
    - **Granular Privilege Planning:** Plan SQL privileges (`SELECT`, `INSERT`, etc.) for roles on tables/views, informing DCL.
    - **Normalization/Integrity:** Proper normalization helps maintain data integrity (a facet of security).
3. **Physical Level (DBMS Implementation - Informed by Modeling):**
    - **GRANT/REVOKE (DAC):** Implement planned access privileges.
    - **Role-Based Access Control (RBAC):** Create roles, grant privileges to roles, assign users to roles.
    - **Stored Procedures:** Encapsulate sensitive DML; grant `EXECUTE` on procedure, not direct table access.
    - **Triggers for Auditing:** Implement triggers to log access to sensitive data.
    - **Encryption:** Plan for column-level or Transparent Data Encryption (TDE) for highly sensitive data.

Considering security at each modeling stage ensures controls align with requirements, protect data, and enforce least privilege.

(Based on: DBMD UNIT - 3 and 4 Notes, p.31-32)

Question 9:

Write short notes on (5x3 = 15 Marks)

a) Clustering

Answer:
Clustering is physically storing related data records close together on disk (ideally same/adjacent

blocks) to minimize Disk I/O and improve query performance for accessing these records together. Data is typically organized by a **clustering key**.

Types: (DBMD UNIT - 3 and 4 Notes, p.28)

1. **Intra-Table Clustering (Clustered Index):** Physical row order within a table matches its clustered index order. A table has at most one. Efficient for range queries on clustering key.
    - **Example:** `Orders` table with clustered index on `OrderDate`; records physically sorted by `OrderDate`.
2. **Inter-Table Clustering (Co-clustering):** Physically interleaving rows from related tables based on a common join key (PK-FK). Speeds up frequent joins.
    - **Example:** Storing `OrderItem` records near their parent `Order` record.

Benefits: Reduced Disk I/O, faster joins (inter-table), efficient range queries (intra-table).
Drawbacks: Slower DML (order maintenance), only one clustered index/table (intra-table).

(Based on: DBMD UNIT - 3 and 4 Notes, p.23, p.28-29)

---

b) De-normalization

Answer:
Denormalization is intentionally introducing controlled redundancy into a relational schema (adding duplicate/grouped data) to improve read performance (query speed) by reducing joins or pre-calculating values. It's the reverse of normalization.

Rationale: (DBMD UNIT - 3 and 4 Notes, p.29-30)
Applied selectively when highly normalized schemas lead to slow queries due to many joins, and indexing/tuning are insufficient.

Techniques: (DBMD UNIT - 3 and 4 Notes, p.30)

1. **Pre-joining Tables:** Adding attributes from "one-side" to "many-side" table (e.g., `DepartmentName` in `EMPLOYEE` table).
2. **Storing Derived/Calculated Values:** Pre-computing expensive values (e.g., `OrderTotal` in `ORDERS` table).
3. **Combining Tables:** Merging tables with 1:1 or tight, frequently accessed 1:N relationships.
4. **Repeating Groups (Limited):** Using multiple columns for fixed, few similar attributes (e.g., `Phone1`, `Phone2`).
5. **Creating Reporting Tables/Data Marts:** Separate, denormalized tables for analytics.

Trade-offs:

- **Benefit:** Faster reads/queries.
- **Cost:** Increased storage, data inconsistency risk (needs careful sync), slower updates, complex DML.

Guideline: Apply judiciously *after* normalization, only for clear performance bottlenecks unresolved by other means.

(Based on: DBMD UNIT - 3 and 4 Notes, p.29-30)

---

c) Database Tuning

Answer:
Database tuning is systematically optimizing database system aspects to improve performance and meet user requirements. It's an iterative process to resolve bottlenecks.

Key Areas: (DBMD UNIT - 3 and 4 Notes, p.22, p.30-31)

1. **Tuning Conceptual Schema (Logical Design):**
   - Normalization/Denormalization evaluation.
   - Vertical/Horizontal Partitioning.
2. **Tuning Queries and Views:**
   - Rewriting inefficient SQL (sargable predicates, simpler logic).
   - Optimizing view definitions.
   - Analyzing execution plans, ensuring up-to-date statistics.
3. **Tuning Physical Design (Indexing and Storage):**
   - Index selection/management (create appropriate, drop unused, rebuild).
   - Clustering decisions.
   - File organization/placement, disk space management.
4. **Tuning Application Code:** Optimizing DB interaction (e.g., batch updates, connection management).
5. **Tuning DBMS Parameters:** Adjusting config (memory, I/O, concurrency). DBMS-specific.
6. **Tuning Hardware and OS:** Ensuring sufficient resources, optimizing OS settings.

Iterative Process: (DBMD UNIT - 3 and 4 Notes, p.22)

1. **Monitor:** Track performance.
2. **Identify Bottlenecks:** Pinpoint poor performance areas.
3. **Diagnose:** Find root cause.
4. **Implement Changes:** Apply tuning measures.

5. **Measure:** Evaluate impact; keep if improved, else revert/retry.

Workload Analysis: (DBMD UNIT - 3 and 4 Notes, p.22)
Critical first step: understand query/update types, frequencies, performance goals, accessed data.

(Based on: DBMD UNIT - 3 and 4 Notes, p.22, p.29-31)
