Database Management Systems: Key Concepts and Application-Level Problems

1 Module 1: Introduction & Entity Relationship (ER) Model

1.1 DBMS Overview

- Software for efficient data storage, retrieval, and management.
- Features: normalization, data integrity, security, backup, structured organization.
- SQL enables database creation, storage, and updates.
- Business-oriented: real-world entity representation, relational databases, structured query languages.
- Scalability, application agility, cybersecurity (e.g., data encryption).

1.2 Data Classification

- **Structured**: organized in relational databases with schemas.
- **Semi-structured**: tagged data with flexible structure (e.g., JSON, XML).
- **Unstructured**: raw data without predefined format (e.g., text, images).

1.3 Data Models

- Types: hierarchical (tree-like), network (many-to-many), relational (tables), ER (conceptual).
- Three-schema architecture: external (user view), conceptual (logical), internal (physical).

1.4 ER Model

- Components: entities (objects), attributes (properties), entity sets, relationships (associations).
- ER diagram: entities (rectangles), attributes (ovals), relationships (diamonds).
- Special types: weak entities (double rectangles), key attributes (underlined), composite (interconnected ovals), multivalued (double ovals), derived (dashed ovals).

- Relationships: cardinalities (1:1, 1:N, N:1, N:M), participation (total/double line, partial/single line).

2 Module 2: Relational Model

2.1 Structure

- Data in tables: rows (records), columns (attributes).
- Primary key: unique row identifier.
- Foreign keys: link related tables.
- Logical vs. physical data separation.

2.2 Integrity Constraints

- **Domain**: permissible attribute values.
- **Key**: unique, non-null primary keys.
- **Referential**: foreign keys reference existing primary keys.

2.3 ER to Relational Conversion

- Regular entities become tables with primary keys.
- Weak entities include identifying entity's primary key as foreign key.
- Relationships:
  - Many-to-many: junction tables.
  - One-to-many: foreign key in ”many” table.
  - One-to-one: foreign key in one table.
- Multivalued attributes: separate tables.

3 Module 3: SQL DML, Physical Data Organization

3.1 SQL Data Manipulation Language (DML)

- **SELECT**: retrieve, filter (WHERE), combine (JOIN).
- **Nested queries**: non-correlated (independent), correlated (dependent).
- **Aggregate functions**: COUNT, SUM, AVG, MIN, MAX with GROUP BY.
- **Views**: virtual tables from stored queries.
- **INSERT, UPDATE, DELETE**: modify data.

3.2 Physical Data Organization

- Key terms: physical/logical records, blocking factor, pinned/unpinned records, heap files.

3.3 Indexing

- Types: single-level, multi-level (B-Trees, B+-Trees).
- Techniques: extendible hashing, grid files.

4 Module 4: Normalization

4.1 Purpose

- Reduce redundancy, prevent anomalies (update, insertion, deletion).

4.2 Functional Dependencies (FD)

- Determinant uniquely determines dependent attributes.
- Armstrong's Axioms: reflexivity, augmentation, transitivity.
- Closure, equivalence, minimal cover.

4.3 Normal Forms

- **1NF**: atomic values.
- **2NF**: no partial dependencies.
- **3NF**: no transitive dependencies.
- **BCNF**: determinants are superkeys.

4.4 Decomposition

- **Lossless join**: reconstruct original relation.
- **Dependency preserving**: maintain FDs.

5 Module 5: Transactions, Concurrency and Recovery, Recent Topics

5.1 Transactions

- Sequence of operations as a single unit.
- **ACID**: Atomicity, Consistency, Isolation, Durability.
- States: Active, Committed, etc.

5.2 Concurrency Control

- Schedules: serial, concurrent.
- Serializability, conflict equivalence.
- Recoverability, cascadeless schedules.
- Locking: 2PL (strict, rigorous, conservative).

5.3 Recovery

- System log for undo/redo.
- Deferred modification.
- Checkpointing.

5.4 Recent Topics: NoSQL Databases

- Flexible schemas, various data models.
- Types:
  - **Key-value** (e.g., Redis): fast, caching.
  - **Document** (e.g., MongoDB): flexible, semi-structured.
  - **Column-family** (e.g., Cassandra): scalable, high availability.
  - **Graph** (e.g., ArangoDB): relationship-oriented.