1 Introduction to Database Design

1.1 Database Design through Entity Relational (ER) Model

The Entity-Relational (ER) model is a way to plan and design a database. It uses diagrams to show how data is organized. An **entity** is like a real-world object, such as a student or a book. Each entity has **attributes**, which are details about it, like a student's name or a book's title. A **relation** is a connection between entities, such as a student enrolling in a course. A **relationship set** is a group of similar connections, like all enrollments in a school. The ER model helps create a clear picture of how data connects, making it easier to build a database (Ramakrishnan & Gehrke, 3rd ed., Ch. 2; Elmasri & Navathe, 6th ed., Ch. 7).

1.2 Keys, Principal Constraints, Weak Entities, Aggregation, Class Hierarchies

A **key** is an attribute (or group of attributes) that uniquely identifies an entity, like a student ID. **Principal constraints** are rules to ensure data stays accurate, such as ensuring every student has a unique ID. **Weak entities** are entities that depend on another entity to exist, like a room in a building (the room needs the building to make sense). **Aggregation** is used to group relationships together to treat them as a single unit, useful for complex connections. **Class hierarchies** organize entities into parent-child relationships, like a general "person" entity with specific types like "student" or "teacher" (Ramakrishnan & Gehrke, 3rd ed., Ch. 2; Elmasri & Navathe, 6th ed., Ch. 8).

1.3 Conceptual Design through ER Model, UML Design

Conceptual design is the process of planning a database using the ER model to map out entities, relationships, and attributes. It's like sketching a blueprint before building. **UML (Unified Modeling Language)** is another tool for designing databases, using diagrams to show how entities and their relationships work together. UML is more detailed and often used in software development, but it complements the ER model for database design (Elmasri & Navathe, 6th ed., Ch. 7; Ramakrishnan & Gehrke, 3rd ed., Ch. 2).

2 Relational Model

2.1 Creating and Modifying Relations using SQL

The **relational model** organizes data into tables (called relations). Each table has rows (data records) and columns (attributes). Using **SQL** (Structured Query Language), you can create tables with commands like `CREATE TABLE` and modify them with `ALTER TABLE`. For example, you can create a table for students with columns for ID and name, then add or remove columns as needed (Ramakrishnan & Gehrke, 3rd ed., Ch. 3; Elmasri & Navathe, 6th ed., Ch. 5).

2.2 Integrity Constraints over Relations

Integrity constraints are rules to keep data accurate. For example, a primary key constraint ensures every row in a table has a unique identifier, like a student ID. Foreign key constraints link tables, ensuring a value in one table matches a value in another. For instance, a course ID in a student enrollment table must exist in the course table (Ramakrishnan & Gehrke, 3rd ed., Ch. 3; Elmasri & Navathe, 6th ed., Ch. 5).

2.3 Transforming ER to Relational Model

To turn an ER model into a relational model, you create a table for each entity and relationship. For example, a ”student” entity becomes a table with columns for its attributes (like ID and name). Relationships, like ”enrolls,” may become tables with foreign keys linking to the related entities. This process ensures the ER design works as a set of tables (Ramakrishnan & Gehrke, 3rd ed., Ch. 2; Elmasri & Navathe, 6th ed., Ch. 9).

2.4 Views and Operations on Views

A **view** is like a virtual table created from a query, showing specific data without storing it separately. For example, a view might show only student names and grades from a larger table. You can query views like regular tables, but some views allow updates, while others are read-only, depending on the query (Ramakrishnan & Gehrke, 3rd ed., Ch. 3; Elmasri & Navathe, 6th ed., Ch. 5).

3 Relational Algebra and Calculus

3.1 Selection, Projection, and Other Set Operations

Relational algebra is a set of operations to manipulate data in tables. **Selection** picks rows that meet a condition, like finding students with grades above 90. **Projection** picks specific columns, like showing only student names. Other **set operations** include union (combining rows from two tables), intersection (finding common rows), and difference (finding rows in one table but not another) (Ramakrishnan & Gehrke, 3rd ed., Ch. 4; Elmasri & Navathe, 6th ed., Ch. 6).

3.2 Joins, Division

A **join** combines two tables based on a condition, like linking students and courses to show who is enrolled where. Types of joins include inner join (only matching rows) and outer join (including unmatched rows). **Division** finds values in one table that are associated with all values in another, like finding students enrolled in every course offered (Ramakrishnan & Gehrke, 3rd ed., Ch. 4; Elmasri & Navathe, 6th ed., Ch. 6).

3.3 Tuple and Domain Relational Calculus

Relational calculus is a way to describe queries using logic instead of operations. **Tuple relational calculus** uses variables representing rows (tuples) to define conditions, like "find all tuples where a student's grade is A." **Domain relational calculus** uses variables for column values, like "find all names where the grade is A." Both are theoretical foundations for SQL (Ramakrishnan & Gehrke, 3rd ed., Ch. 4; Elmasri & Navathe, 6th ed., Ch. 6).

4 SQL Queries

4.1 Basic SQL Queries

Basic SQL queries retrieve data using commands like `SELECT`, `FROM`, and `WHERE`. For example, `SELECT name FROM students WHERE grade = 'A'` finds names of students with an A grade. These queries are the foundation of working with databases (Ramakrishnan & Gehrke, 3rd ed., Ch. 5; Elmasri & Navathe, 6th ed., Ch. 5).

4.2 Nested SQL Queries

Nested SQL queries are queries inside other queries. For example, you might find students whose grades are higher than the average grade by nesting a query to calculate the average inside the main query. These allow complex data retrieval (Ramakrishnan & Gehrke, 3rd ed., Ch. 5; Elmasri & Navathe, 6th ed., Ch. 5).

4.3 Aggregate and Join Operations

Aggregate operations summarize data, like `COUNT` (counting rows), `SUM`, `AVG`, `MAX`, or `MIN`. For example, `SELECT AVG(grade) FROM students` finds the average grade. **Join operations** in SQL combine tables, like `SELECT students.name, courses.title FROM students JOIN courses ON students.courseID = courses.id` to show student-course pairs (Ramakrishnan & Gehrke, 3rd ed., Ch. 5; Elmasri & Navathe, 6th ed., Ch. 5).

4.4 Complex Integrity and Triggers

Complex integrity constraints enforce rules beyond simple keys, like ensuring a student's grade is between 0 and 100. **Triggers** are actions that automatically run when data changes, like updating a student's status when their grade is entered. Triggers help maintain data consistency (Ramakrishnan & Gehrke, 3rd ed., Ch. 5; Elmasri & Navathe, 6th ed., Ch. 5).

5 Transaction Management

5.1 ACID Properties of Transaction

A **transaction** is a sequence of operations treated as one unit, like transferring money between bank accounts. **ACID** stands for:

- **Atomicity**: All operations complete, or none do.
- **Consistency**: The database stays valid after the transaction.
- **Isolation**: Transactions don't interfere with each other.
- **Durability**: Completed transactions are saved, even if the system crashes.

These ensure reliable database operations (Ramakrishnan & Gehrke, 3rd ed., Ch. 16; Elmasri & Navathe, 6th ed., Ch. 21).

5.2 Serializability, Lock-Based Concurrency Control, Deadlocks, Performance of Locking

Serializability ensures transactions run as if they happened one after another, avoiding conflicts. **Lock-based concurrency control** uses locks to prevent multiple transactions from changing the same data at once. **Deadlocks** happen when transactions wait for each other's locks, freezing the system. **Performance of locking** depends on how locks are managed—too many locks slow things down, but too few cause errors (Ramakrishnan & Gehrke, 3rd ed., Ch. 16; Elmasri & Navathe, 6th ed., Ch. 22).

5.3 Deadlock Prevention, Timestamp-Based Concurrency Control

Deadlock prevention stops deadlocks by setting rules, like making transactions wait or abort if they might cause a deadlock. **Timestamp-based concurrency control** assigns each transaction a timestamp and uses it to decide the order of operations, avoiding conflicts without locks (Ramakrishnan & Gehrke, 3rd ed., Ch. 16; Elmasri & Navathe, 6th ed., Ch. 22).

6 Schema Refinement and Normal Forms

6.1 Introduction to Schema Refinement and Problems Caused by Redundancy

Schema refinement improves database design to avoid problems like **redundancy**, where the same data is stored multiple times, wasting space and risking errors. For example, repeating a student's name in multiple tables can lead to inconsistencies if the name changes (Ramakrishnan & Gehrke, 3rd ed., Ch. 19; Elmasri & Navathe, 6th ed., Ch. 14).

6.2 Functional Dependencies, Different Normal Forms

Functional dependencies are rules where one attribute determines another, like a student ID determining their name. **Normal forms** are standards to organize tables and reduce redundancy:

- **1NF** (First Normal Form): All attributes are simple (no lists in a column).
- **2NF**: No partial dependency (non-key attributes depend on the whole key).
- **3NF**: No transitive dependency (non-key attributes don't depend on other non-key attributes).
- **BCNF** (Boyce-Codd Normal Form): A stricter version of 3NF.

Each form builds on the previous one to make the database more efficient (Ramakrishnan & Gehrke, 3rd ed., Ch. 19; Elmasri & Navathe, 6th ed., Ch. 14).

6.3 Lossless Join Decomposition and Dependency Preserving Decomposition

Lossless join decomposition splits a table into smaller tables that can be rejoined without losing data. **Dependency preserving decomposition** ensures that functional dependencies (rules) are still valid after splitting. These techniques help create efficient, non-redundant databases while keeping all data and rules intact (Ramakrishnan & Gehrke, 3rd ed., Ch. 19; Elmasri & Navathe, 6th ed., Ch. 14).