DATABASE MANAGEMENT SYSTEMS

Complete Study Guide for GATE Examination

Topics Covered:

✓ ER Model & Relational Model
✓ Normalization & Functional Dependencies
✓ SQL & Relational Algebra
✓ Transactions & Concurrency Control
✓ Indexing & B-Trees

✓ 100+ GATE Questions with Solutions

TABLE OF CONTENTS

PART I: DATABASE FUNDAMENTALS

Chapter 1: Introduction to Databases

1.1 What is a Database?

A database is a collection of inter-related data organized in a structured manner that represents aspects of the real world. It allows for efficient data storage, retrieval, and manipulation.

Key Characteristics:

- Represents real-world entities and their relationships
- Data is logically related and organized
- Supports efficient querying and updates
- Maintains data integrity and consistency
- Provides concurrent access control

1.2 Evolution from File Systems

Before databases, organizations used file systems. However, they had critical limitations:

Problems with File Systems:

1. Data Redundancy: Same information stored multiple times in different files
2. Data Inconsistency: Updates in one file don't reflect in others
3. Difficulty in Data Access: Need to write new programs for each query
4. Integrity Problems: Hard to enforce constraints across files
5. Atomicity Issues: Partial updates can leave data inconsistent
6. Concurrent Access Problems: Race conditions and conflicts
7. Security Issues: Difficult to provide granular access control

Databases solved these problems through:

- Centralized data storage and management
- Built-in integrity constraints
- ACID properties for transactions
- Concurrent access control mechanisms
- Sophisticated security features

1.3 Database Terminology

Database Schema:

- The skeletal design or blueprint of the database
- Defines structure without actual data
- Specifies tables, attributes, relationships, and constraints
- Remains relatively stable over time

Database Instance:

- The actual data stored in database at a particular moment
- Changes frequently as data is inserted, updated, or deleted
- Must conform to the schema definition

Data Constraints:

- Rules that restrict the kind of data that can be stored
- Types: Domain, Entity Integrity, Referential Integrity, Key constraints
- Enforced automatically by DBMS

Metadata/Data Dictionary:

- Data about data
- Combination of schema and constraints
- Describes structure, types, relationships, and meanings
- Used by DBMS to manage and interpret data

Query:

- Request to retrieve or manipulate data
- Written in query language (e.g., SQL)
- Processed by query processor and optimizer

Data Manipulation Operations:

- INSERT: Add new data to database
- UPDATE: Modify existing data
- DELETE: Remove data from database
- SELECT: Retrieve data from database

Database vs Instance Example

Schema (structure):

```
Student(Roll_No: Integer, Name: String, Age: Integer, Department:
String)
```

Instance (data at time T1):

```
101 | John | 20 | CS
102 | Mary | 21 | EE
```

Instance (data at time T2 after insert):

```
101 | John | 20 | CS
102 | Mary | 21 | EE
103 | Bob | 19 | ME
```

1.4 Advantages of Database Systems

8. Data Sharing
    - Multiple users can access data simultaneously
    - Controlled access through permissions
    - Different views for different users
9. Reduced Data Redundancy
    - Data stored once and referenced multiple times
    - Saves storage space
    - Reduces inconsistency risks
10. Data Integrity and Consistency
    - Enforces constraints automatically
    - Maintains data accuracy
    - Prevents invalid data entry
11. Data Security
    - User authentication and authorization
    - Granular access control
    - Audit trails
12. Backup and Recovery
    - Automated backup mechanisms
    - Point-in-time recovery
    - Protection against data loss
13. Concurrency Control
    - Multiple users can work simultaneously
    - Prevents conflicts and inconsistencies
    - Maintains ACID properties
14. Data Abstraction
    - Hides complexity from users
    - Provides different views at different levels
    - Simplifies application development

1.5 Disadvantages of Database Systems

- High Initial Cost: Expensive software and hardware
- Complexity: Requires specialized knowledge and training
- Performance Overhead: May be slower for simple operations
- Single Point of Failure: If database fails, entire system affected
- Scalability Challenges: Difficult to scale horizontally

Chapter 2: Types of Databases

2.1 Hierarchical Database

Organizes data in tree-like structure with parent-child relationships.

Characteristics:

- Each parent can have multiple children
- Each child has exactly one parent
- Root node has no parent
- Relationships are 1:N (one-to-many)

Advantages:

- Simple and easy to understand
- Fast data access for hierarchical relationships
- Data integrity through parent-child links

Disadvantages:

- Inflexible structure
- Difficulty representing M:N relationships
- Data redundancy if same entity appears at multiple levels

Examples:

- File systems (folders and files)
- Organizational charts
- XML databases
- Windows Registry

2.2 Network Database

Extension of hierarchical model allowing multiple parent-child relationships.

Characteristics:

- Child can have multiple parents
- Represented as graph structure
- Supports M:N relationships
- More flexible than hierarchical model

Advantages:

- Can represent complex relationships
- Better than hierarchical for many-to-many
- Efficient for networked data

Disadvantages:

- Complex structure

- Difficult to design and maintain
- Navigational access (must follow pointers)

Example: CODASYL database systems

2.3 Object-Oriented Database

Stores data as objects, similar to object-oriented programming.

Characteristics:

- Data represented as objects with attributes and methods
- Supports inheritance and polymorphism
- Can store complex data types (multimedia)
- Objects can contain other objects

Advantages:

- Natural representation of real-world entities
- Supports complex data types
- Reusability through inheritance
- Good for multimedia applications

Disadvantages:

- Lack of standardization
- Steep learning curve
- Limited adoption compared to relational

Examples:

- CAD/CAM systems
- Multimedia databases
- GIS (Geographic Information Systems)

2.4 Relational Database (Most Widely Used)

Organizes data in tables (relations) with rows and columns.

Characteristics:

- Data stored in tables (relations)
- Each row is a tuple (record)
- Each column is an attribute (field)
- Primary keys uniquely identify rows
- Foreign keys establish relationships between tables
- Based on mathematical set theory and relational algebra

Why Most Popular:

- Simple and intuitive structure
- Powerful query language (SQL)

- Strong theoretical foundation
- ACID properties support
- Mature ecosystem and tools
- Widely standardized

Examples:

- MySQL
- PostgreSQL
- Oracle Database
- Microsoft SQL Server
- SQLite

Relational Database Example

Student Table:

```
Roll_No | Name    | Age | Dept_ID
101     | John    | 20  | D1
102     | Mary    | 21  | D2
```

Department Table:

```
Dept_ID | Dept_Name
D1      | Computer Science
D2      | Electrical Engg
```

Relationship: Student.Dept_ID references Department.Dept_ID

Chapter 3: Database Management System (DBMS)

3.1 What is DBMS?

DBMS is software (or collection of programs) that manages and controls access to the database. It acts as an interface between users and the underlying data.

Functions of DBMS:

- Data Definition: Define schemas and constraints
- Data Manipulation: Insert, update, delete, query
- Data Security: Authentication and authorization
- Data Integrity: Enforce constraints
- Concurrency Control: Handle simultaneous access
- Backup and Recovery: Protect against data loss
- Performance Optimization: Query optimization

3.2 DBMS Users

1. Application Programmers:
    - Write programs that interact with database
    - Use programming language APIs (JDBC, ODBC)
    - Develop database applications
    - Create user interfaces
2. Database Administrators (DBA):
    - Manage entire DBMS environment
    - Grant/revoke user permissions
    - Perform backup and recovery
    - Monitor performance and optimize
    - Design and maintain schemas
    - Ensure security and data integrity
3. End Users:
    - Interact with database through applications
    - Execute queries and reports
    - Don't need technical DBMS knowledge
    - Types:
        - Casual users: Occasional access
        - Naive users: Use predefined transactions
        - Sophisticated users: Use query tools directly

3.3 DBMS Architecture

3.3.1 Single-Tier Architecture

Database and DBMS reside on same system as user interface.

Characteristics:

- No network involved
- Direct file access
- Simple deployment
- Good for standalone applications

Limitations:

- No sharing between users
- No concurrent access
- Limited scalability

Use Cases:

- Personal databases
- Mobile apps with local database
- Development and testing

3.3.2 Two-Tier Architecture (Client-Server)

Client sends requests to server, which processes and returns results.

Components:

- Tier 1 (Client): User interface and application logic
- Tier 2 (Server): Database and DBMS

Advantages:

- Better security (centralized)
- Easier maintenance
- Data sharing possible
- Clear separation of concerns

Disadvantages:

- Performance degradation with many clients
- Business logic mixed with presentation
- Difficult to modify

Examples:

- Desktop applications with central database
- Simple web applications

3.3.3 Three-Tier Architecture

Adds middle layer between client and database.

Components:

- Tier 1 (Presentation): User interface
- Tier 2 (Application Server): Business logic
- Tier 3 (Database Server): Data storage and management

Advantages:

- Better scalability
- Improved security (database not directly accessible)
- Easier to modify and maintain
- Can distribute load across multiple servers
- Reusable business logic

Best For:

- Large web applications
- Enterprise systems
- E-commerce platforms

Three-Tier Example

Online Shopping System:

  - Tier 1: Web browser showing products
  - Tier 2: Web server handling cart logic, payment processing
  - Tier 3: Database storing product catalog, orders, users

3.4 DBMS Schemas (Three-Level Architecture)

ANSI-SPARC architecture divides DBMS into three abstract levels:

3.4.1 Physical/Internal Schema

Lowest level - describes physical storage.

- How data is stored in blocks on disk
- Storage structures (sequential, indexed, hash)
- Access paths and indexes
- File organization
- Data compression and encryption

Managed by: DBMS and DBA

3.4.2 Logical/Conceptual Schema

Middle level - describes logical structure.

- What data is stored (tables, attributes, data types)
- Relationships between data
- Constraints and integrity rules
- Security and authorization

Managed by: DBA and Programmers

Most important level for database design

3.4.3 External/View Schema

Highest level - describes how users see data.

- Different views for different users
- Hides unnecessary details
- Provides customized perspectives
- Multiple external schemas possible

Managed by: Application developers

End users interact at this level

Three-Schema Example

Physical Schema:

- ○ Student table stored in file student.dat using B-tree index on Roll_No

Logical Schema:

- ○ Student(Roll_No: int PK, Name: varchar(50), Age: int, Dept: varchar(20))

External Schema (View for faculty):

- ○ CREATE VIEW FacultyStudentView AS SELECT Roll_No, Name FROM Student

3.5 Data Independence

Ability to change schema at one level without affecting schema at next higher level.

3.5.1 Physical Data Independence

- Change physical storage without changing logical schema
- Examples:
  - Changing from sequential to indexed file
  - Adding new index
  - Changing storage device
  - Modifying data compression method
- Easier to achieve than logical independence

3.5.2 Logical Data Independence

- Change logical schema without changing external views
- Examples:
  - Adding new attribute to table
  - Adding new table
  - Merging tables
- Harder to achieve than physical independence

IMPORTANT: Data independence is crucial for database evolution. It allows systems to adapt to changing requirements without disrupting existing applications.

PART II: DATA LANGUAGES

Chapter 4: SQL and Data Languages

4.1 Types of Data Languages

SQL (Structured Query Language) is divided into four sublanguages:

4.1.1 Data Definition Language (DDL)

Defines database structure and schema.

Commands:

- CREATE: Create database objects (tables, indexes, views)
- ALTER: Modify structure of existing objects
- DROP: Delete database objects
- TRUNCATE: Remove all records from table (faster than DELETE)
- RENAME: Rename database objects
- COMMENT: Add comments to data dictionary

DDL Examples

```
CREATE TABLE Student (
    Roll_No INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Age INT CHECK (Age >= 17),
    Dept VARCHAR(20)
);
```

```
ALTER TABLE Student ADD Email VARCHAR(100);
ALTER TABLE Student MODIFY Age INT CHECK (Age >= 16);
ALTER TABLE Student DROP COLUMN Email;
```

```
TRUNCATE TABLE Student;  -- Removes all rows
DROP TABLE Student;      -- Deletes entire table
```

4.1.2 Data Manipulation Language (DML)

Manipulates data within database objects.

Commands:

- SELECT: Retrieve data
- INSERT: Add new records
- UPDATE: Modify existing records

- DELETE: Remove records
- MERGE: Combine insert and update

DML Examples

```
-- INSERT
INSERT INTO Student VALUES (101, 'John', 20, 'CS');
INSERT INTO Student (Roll_No, Name) VALUES (102, 'Mary');
```

```
-- UPDATE
UPDATE Student SET Age = 21 WHERE Roll_No = 101;
UPDATE Student SET Dept = 'CS' WHERE Age > 20;
```

```
-- DELETE
DELETE FROM Student WHERE Roll_No = 101;
DELETE FROM Student WHERE Age < 18;
```

```
-- SELECT
SELECT * FROM Student;
SELECT Name, Age FROM Student WHERE Dept = 'CS';
SELECT DISTINCT Dept FROM Student;
```

4.1.3 Data Control Language (DCL)

Controls access to data.

Commands:

- GRANT: Give permissions to users
- REVOKE: Remove permissions from users

DCL Examples

```
-- Grant SELECT permission
GRANT SELECT ON Student TO user1;
```

```
-- Grant multiple permissions
GRANT SELECT, INSERT, UPDATE ON Student TO user2;
```

```
-- Revoke permissions
REVOKE INSERT ON Student FROM user2;
```

```
-- Grant all privileges
GRANT ALL PRIVILEGES ON Student TO admin_user;
```

4.1.4 Transaction Control Language (TCL)

Manages transactions to ensure ACID properties.

Commands:

- COMMIT: Save all changes permanently
- ROLLBACK: Undo all changes since last COMMIT
- SAVEPOINT: Set a point to rollback to

TCL Examples

```
BEGIN TRANSACTION;
    INSERT INTO Student VALUES (101, 'John', 20, 'CS');
    SAVEPOINT sp1;
    UPDATE Student SET Age = 21 WHERE Roll_No = 101;
    SAVEPOINT sp2;
    DELETE FROM Student WHERE Roll_No = 100;
    ROLLBACK TO sp2;  -- Undo delete
COMMIT;  -- Save insert and update
```

4.2 Advanced SQL Queries

4.2.1 SELECT Statement

Basic Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column
HAVING group_condition
ORDER BY column [ASC|DESC];
```

SELECT DISTINCT - Remove duplicates:

```
SELECT DISTINCT Dept FROM Student;
```

SELECT with Aliases:

```
SELECT Name AS StudentName, Age AS StudentAge FROM Student;
```

4.2.2 WHERE Clause

Filters rows based on conditions.

Comparison Operators:

- = (equal), <> or != (not equal)
- < (less than), > (greater than)
- <= (less than or equal), >= (greater than or equal)

Logical Operators:

- AND, OR, NOT

WHERE Examples

```
-- Single condition
SELECT * FROM Student WHERE Age > 20;
```

```
-- Multiple conditions
SELECT * FROM Student WHERE Age > 18 AND Dept = 'CS';
SELECT * FROM Student WHERE Dept = 'CS' OR Dept = 'EE';
```

```
-- NOT operator
SELECT * FROM Student WHERE NOT Dept = 'CS';
```

4.2.3 LIKE Operator

Pattern matching in WHERE clause.

- % - Matches zero or more characters

- _ - Matches exactly one character

LIKE Examples

```
-- Names starting with 'J'
SELECT * FROM Student WHERE Name LIKE 'J%';
```

```
-- Names ending with 'n'
SELECT * FROM Student WHERE Name LIKE '%n';
```

```
-- Names with exactly 4 characters
SELECT * FROM Student WHERE Name LIKE '____';
```

```
-- Names with 'ar' anywhere
SELECT * FROM Student WHERE Name LIKE '%ar%';
```

4.2.4 IN Operator

Check if value matches any in a list.

IN Examples

```
SELECT * FROM Student WHERE Dept IN ('CS', 'EE', 'ME');
SELECT * FROM Student WHERE Age IN (18, 19, 20);
```

```
-- Equivalent to:
SELECT * FROM Student WHERE Dept = 'CS' OR Dept = 'EE' OR Dept = 'ME';
```

4.2.5 BETWEEN Operator

Select values within a range (inclusive).

BETWEEN Examples

```
SELECT * FROM Student WHERE Age BETWEEN 18 AND 21;
```

```
-- Equivalent to:
SELECT * FROM Student WHERE Age >= 18 AND Age <= 21;
```

4.3 Aggregate Functions

Perform calculations on multiple rows and return single value.

Common Aggregate Functions:

- COUNT() - Count number of rows
- SUM() - Sum of numeric values
- AVG() - Average of numeric values
- MIN() - Minimum value
- MAX() - Maximum value

Aggregate Function Examples

```
-- Count total students
SELECT COUNT(*) FROM Student;
```

```
-- Count students in CS department
SELECT COUNT(*) FROM Student WHERE Dept = 'CS';
```

```
-- Count distinct departments
SELECT COUNT(DISTINCT Dept) FROM Student;
```

```
-- Average age
SELECT AVG(Age) FROM Student;
```

```
-- Minimum and maximum age
SELECT MIN(Age), MAX(Age) FROM Student;
```

IMPORTANT: COUNT(*) counts all rows including NULLs. COUNT(column) counts only non-NULL values.

4.4 GROUP BY Clause

Groups rows with same values in specified columns.

Often used with aggregate functions.

GROUP BY Examples

```
-- Count students in each department
SELECT Dept, COUNT(*) AS StudentCount
FROM Student
GROUP BY Dept;
```

```
-- Average age per department
```

```
SELECT Dept, AVG(Age) AS AvgAge
FROM Student
GROUP BY Dept;
```

```
-- Multiple columns in GROUP BY
SELECT Dept, Age, COUNT(*)
FROM Student
GROUP BY Dept, Age;
```

4.5 HAVING Clause

Filters groups created by GROUP BY (cannot use WHERE for this).

HAVING Examples

```
-- Departments with more than 10 students
SELECT Dept, COUNT(*) AS StudentCount
FROM Student
GROUP BY Dept
HAVING COUNT(*) > 10;
```

```
-- Departments with average age > 20
SELECT Dept, AVG(Age)
FROM Student
GROUP BY Dept
HAVING AVG(Age) > 20;
```

NOTE: WHERE filters rows before grouping. HAVING filters groups after grouping.

4.6 ORDER BY Clause

Sorts result set.

- ASC - Ascending order (default)
- DESC - Descending order

ORDER BY Examples

```
-- Sort by name ascending
SELECT * FROM Student ORDER BY Name;
SELECT * FROM Student ORDER BY Name ASC;  -- Same as above
```

```
-- Sort by age descending
SELECT * FROM Student ORDER BY Age DESC;
```

```
-- Multiple column sorting
SELECT * FROM Student ORDER BY Dept ASC, Age DESC;
```

4.7 SQL JOINS

Combine rows from two or more tables based on related columns.

4.7.1 INNER JOIN

Returns rows when there is a match in both tables.

INNER JOIN Example

Tables:

```
Student(Roll_No, Name, Dept_ID)
Department(Dept_ID, Dept_Name)
```

```
SELECT Student.Name, Department.Dept_Name
FROM Student
INNER JOIN Department
ON Student.Dept_ID = Department.Dept_ID;
```

Returns only students who have a matching department.

4.7.2 LEFT (OUTER) JOIN

Returns all rows from left table, matching rows from right table.

NULL for right table columns if no match.

LEFT JOIN Example

```
SELECT Student.Name, Department.Dept_Name
FROM Student
LEFT JOIN Department
ON Student.Dept_ID = Department.Dept_ID;
```

Returns all students, even those without matching department (Dept_Name will be NULL).

4.7.3 RIGHT (OUTER) JOIN

Returns all rows from right table, matching rows from left table.

NULL for left table columns if no match.

RIGHT JOIN Example

```
SELECT Student.Name, Department.Dept_Name
FROM Student
RIGHT JOIN Department
ON Student.Dept_ID = Department.Dept_ID;
```

Returns all departments, even those with no students (Name will be NULL).

4.7.4 FULL (OUTER) JOIN

Returns all rows when there is a match in either table.
NULL where there's no match.

FULL JOIN Example

```
SELECT Student.Name, Department.Dept_Name
FROM Student
FULL OUTER JOIN Department
ON Student.Dept_ID = Department.Dept_ID;
```

Returns all students and all departments. NULL where no match.

4.7.5 CROSS JOIN (Cartesian Product)

Returns Cartesian product of both tables.
Each row from first table combined with each row from second.

CROSS JOIN Example

```
SELECT * FROM Student CROSS JOIN Department;
```

If Student has 100 rows and Department has 5 rows, result has 500 rows.

4.8 Set Operations

Combine results of multiple SELECT statements.

4.8.1 UNION

Combines results and removes duplicates.

- Tables must have same number of columns
- Corresponding columns must have compatible data types

UNION Example

```
SELECT Name FROM Student WHERE Dept = 'CS'
UNION
SELECT Name FROM Student WHERE Age > 20;
```

Returns names of students in CS or Age > 20, removing duplicates.

4.8.2 UNION ALL

Combines results keeping all duplicates.

UNION ALL Example

```
SELECT Name FROM Student WHERE Dept = 'CS'
UNION ALL
SELECT Name FROM Student WHERE Age > 20;
```

Returns all matching rows, including duplicates.

4.8.3 INTERSECT

Returns only rows common to both queries.

INTERSECT Example

```
SELECT Name FROM Student WHERE Dept = 'CS'
INTERSECT
SELECT Name FROM Student WHERE Age > 20;
```

Returns students who are both in CS AND have Age > 20.

4.8.4 MINUS (or EXCEPT)

Returns rows from first query not in second query.

MINUS Example

```
SELECT Name FROM Student WHERE Dept = 'CS'
```

```
MINUS
SELECT Name FROM Student WHERE Age > 20;
```

Returns CS students who do NOT have Age > 20.

4.9 Nested Queries (Subqueries)

Query within another query.

4.9.1 Subquery in WHERE Clause

Subquery Examples

```
-- Find students in departments with more than 50 students
SELECT Name FROM Student
WHERE Dept_ID IN (
    SELECT Dept_ID FROM Student
    GROUP BY Dept_ID
    HAVING COUNT(*) > 50
);
```

```
-- Find students older than average age
SELECT Name, Age FROM Student
WHERE Age > (SELECT AVG(Age) FROM Student);
```

4.9.2 Correlated Subquery

Subquery references column from outer query.

Executed once for each row in outer query.

Correlated Subquery Example

```
-- Find students older than average age in their department
SELECT s1.Name, s1.Age, s1.Dept
FROM Student s1
WHERE s1.Age > (
    SELECT AVG(s2.Age)
    FROM Student s2
    WHERE s2.Dept = s1.Dept
);
```

4.9.3 EXISTS Operator

Returns TRUE if subquery returns any rows.

EXISTS Example

```
-- Find departments that have at least one student
SELECT Dept_Name FROM Department d
WHERE EXISTS (
    SELECT 1 FROM Student s
    WHERE s.Dept_ID = d.Dept_ID
);
```

PART III: ER MODEL AND DESIGN

Chapter 5: Entity-Relationship Model

PART IV: NORMALIZATION

Chapter 10: Normalization and Normal Forms

10.1 Introduction to Normalization

Normalization is the process of organizing data to minimize redundancy and dependency.

Goals of Normalization:

- Eliminate redundant data
- Ensure data dependencies make sense
- Reduce space required to store data
- Avoid insertion, deletion, and update anomalies

10.2 Anomalies in Unnormalized Databases

10.2.1 Insertion Anomaly

Cannot insert data without presence of other data.

Insertion Anomaly

Table: Student_Course(Roll_No, Name, Course_ID, Course_Name, Instructor)

Problem: Cannot add new course without enrolling a student.

10.2.2 Deletion Anomaly

Deleting data causes unintended loss of other data.

Deletion Anomaly

Same table as above.

Problem: If last student drops a course, we lose course information.

10.2.3 Update Anomaly

Updating data requires multiple changes, risk of inconsistency.

Update Anomaly

Same table as above.

Problem: If instructor changes, must update all rows with that course.

10.3 Functional Dependencies

Functional Dependency (FD): X → Y means value of X uniquely determines value of Y.

Notation: X → Y (X functionally determines Y)

- X is determinant
- Y is dependent

Functional Dependency Examples

Student(Roll_No, Name, Dept, Advisor)

Functional Dependencies:

- ○ Roll_No → Name (Roll number determines name)
- ○ Roll_No → Dept (Roll number determines department)
- ○ Roll_No → Advisor (Roll number determines advisor)
- ○ Roll_No → Name, Dept, Advisor (Combined)

10.3.1 Types of Functional Dependencies

1. Trivial FD:
   - X → Y where Y ⊆ X
   - Example: {Roll_No, Name} → Roll_No
   - Always holds true
2. Non-Trivial FD:
   - X → Y where Y ⊄ X
   - Actually constrains the data
3. Completely Non-Trivial FD:
   - X → Y where X ∩ Y = ∅
   - No overlap between X and Y
4. Partial Dependency:
   - Non-prime attribute depends on part of candidate key
   - Example: {Roll_No, Course_ID} → Student_Name
   - Student_Name depends only on Roll_No, not Course_ID
5. Transitive Dependency:
   - X → Y and Y → Z, then X → Z
   - Non-prime attribute depends on another non-prime attribute

10.4 Armstrong's Axioms

Inference rules to derive new functional dependencies.

Primary Axioms:

15. Reflexivity: If Y ⊆ X, then X → Y
16. Augmentation: If X → Y, then XZ → YZ
17. Transitivity: If X → Y and Y → Z, then X → Z

Secondary Rules (derived from primary):

18. Union: If X → Y and X → Z, then X → YZ
19. Decomposition: If X → YZ, then X → Y and X → Z
20. Pseudo-transitivity: If X → Y and YZ → W, then XZ → W
21. Composition: If X → Y and A → B, then XA → YB

Armstrong's Axioms Application

Given: A → B, B → C

Derive: A → C

Proof:

- A → B (Given)
- B → C (Given)
- A → C (By Transitivity)

10.5 Closure of Attributes

X+ = Set of all attributes functionally determined by X

Closure Calculation

R(A, B, C, D, E)

FDs: A → B, B → C, CD → E

Find: {A}+

Solution:

- {A}+ = {A} (initially)
- A → B, so {A}+ = {A, B}
- B → C, so {A}+ = {A, B, C}
- Cannot use CD → E (don't have D)
- Final: {A}+ = {A, B, C}

10.6 Normal Forms

10.6.1 First Normal Form (1NF)

Requirements:

- All attributes must contain atomic values (no multi-valued or composite attributes)
- No repeating groups
- Each row must be unique

1NF Normalization

Unnormalized:

```
Student(Roll_No, Name, Phone_Numbers)
101 | John | {123-4567, 987-6543}
```

Normalized to 1NF:

```
Student(Roll_No, Name)
101 | John

Student_Phone(Roll_No, Phone_Number)
101 | 123-4567
101 | 987-6543
```

10.6.2 Second Normal Form (2NF)

Requirements:

- Must be in 1NF
- No partial dependency (no non-prime attribute should depend on part of candidate key)

2NF Normalization

Not in 2NF:

```
Enrollment(Roll_No, Course_ID, Student_Name, Course_Name, Grade)
Primary Key: {Roll_No, Course_ID}
FDs: Roll_No → Student_Name, Course_ID → Course_Name
```

Problem: Student_Name depends only on Roll_No (partial dependency)

Normalized to 2NF:

```
Student(Roll_No, Student_Name)
Course(Course_ID, Course_Name)
Enrollment(Roll_No, Course_ID, Grade)
```

10.6.3 Third Normal Form (3NF)

Requirements:

- Must be in 2NF
- No transitive dependency (no non-prime attribute should depend on another non-prime attribute)

3NF Normalization

Not in 3NF:

```
Student(Roll_No, Name, Dept_ID, Dept_Name, Dept_Building)
Primary Key: Roll_No
FDs: Roll_No → Dept_ID, Dept_ID → Dept_Name, Dept_ID → Dept_Building
```

Problem: Dept_Name and Dept_Building depend on Dept_ID (transitive)

Normalized to 3NF:

```
Student(Roll_No, Name, Dept_ID)
Department(Dept_ID, Dept_Name, Dept_Building)
```

10.6.4 Boyce-Codd Normal Form (BCNF)

Stronger version of 3NF.

Requirements:

- Must be in 3NF
- For every FD X → Y, X must be a super key

IMPORTANT: Every relation in BCNF is also in 3NF, but not vice versa.

BCNF Example

R(A, B, C)

FDs: AB → C, C → B

Candidate Keys: {AB}, {AC}

Check BCNF:

- AB → C: AB is super key ✓
- C → B: C is NOT super key ✗

Not in BCNF!

Decompose to BCNF:

```
R1(C, B)  -- C → B
R2(A, C)  -- AC is key
```

10.6.5 Fourth Normal Form (4NF)

Requirements:

- Must be in BCNF
- No multi-valued dependencies

Multi-valued Dependency (MVD): X ↠ Y

- For each value of X, there's a set of values for Y
- Independent of other attributes

4NF Example

Student_Hobby_Mobile(Student, Hobby, Mobile)

MVDs: Student ↠ Hobby, Student ↠ Mobile

Problem: Hobbies and mobiles are independent but stored together

Decompose to 4NF:

```
Student_Hobby(Student, Hobby)
Student_Mobile(Student, Mobile)
```

10.6.6 Fifth Normal Form (5NF)

Requirements:

- Must be in 4NF
- No join dependency
- Cannot be decomposed into smaller relations without loss of information

Also called Project-Join Normal Form (PJNF)

PART V: GATE QUESTIONS WITH SOLUTIONS

Chapter 20: GATE Questions - ER Model

GATE Question 1

Consider an ER model with entities E1 and E2. E1 has attributes {A, B, C} and E2 has attributes {D, E, F}. If E1 and E2 have a many-to-one relationship R with total participation from E1 side, what is the minimum number of tables required to represent this model?

✓Answer: 2 tables

Explanation:

In many-to-one relationship with total participation from many side (E1), we can combine E1 and relationship R into single table. So we need: E1R(A, B, C, D) and E2(D, E, F). Total = 2 tables.

GATE Question 2

In an ER diagram, a weak entity set can be identified by:

✓Answer: (A) Double rectangle

Explanation:

Weak entity sets are represented by double rectangles in ER diagrams. They don't have a primary key of their own and depend on a strong entity set. The discriminator (partial key) is shown with dashed underline.

GATE Question 3

What is the minimum number of tables required to represent the following ER model: Entity E1(a1, a2) has one-to-one relationship R1 with E2(b1, b2) with partial participation on both sides?

✓Answer: 2 tables

Explanation:

For one-to-one with both partial participation, we can combine either E1 with R1 or E2 with R1. Optimal: E1R1(a1, a2, b1) and E2(b1, b2), giving us 2 tables minimum.

GATE Question 4

In a university database, Student can enroll in multiple Courses and each Course can have multiple Students. A Student's grade in a Course should be stored. Which of the following is correct?

✓Answer: Grade should be an attribute of the Enrollment relationship

Explanation:

Grade depends on both Student AND Course (it's the grade of a specific student in a specific course). Therefore, it must be an attribute of the many-to-many relationship between Student and Course, not of either entity alone.

Chapter 21: GATE Questions - Keys and Dependencies

GATE Question 5

Consider relation R(A, B, C, D, E) with FDs: A → B, BC → E, ED → A. What are the candidate keys?

✅Answer: {A, C, D} and {B, C, D} and {C, D, E}

Explanation:

To find candidate keys, identify attributes that never appear on RHS (must be in every key): C, D. Then find minimal sets containing C, D that determine all attributes: {ACD}+ = ABCDE ✓, {BCD}+ = ABCDE ✓, {CDE}+ = ABCDE ✓

GATE Question 6

If a relation R has n attributes and candidate key has k attributes, how many super keys are possible?

✅Answer: 2^(n-k)

Explanation:

Super keys are formed by adding any combination of the remaining (n-k) attributes to the candidate key. Each of these (n-k) attributes can either be included or not included, giving 2^(n-k) possibilities.

GATE Question 7

Consider R(A,B,C,D,E,F) with FD: ABC → DEF. For relation to be in 2NF:

✅Answer: There should be no partial dependency

Explanation:

2NF requires no partial dependency, meaning no non-prime attribute should depend on proper subset of candidate key. Here, if ABC is the only candidate key, then A, B, C are prime attributes and D, E, F are non-prime. If any of DEF depends on just AB or just BC, it violates 2NF.

Chapter 22: GATE Questions - Normalization

GATE Question 8

Relation R(A,B,C,D) with FDs: A → B, B → C, C → D, D → A. What is the highest normal form?

✓Answer: BCNF

Explanation:

Candidate keys: A, B, C, D (all are candidate keys since they form a cycle). All attributes are prime. For BCNF, every determinant must be super key. All FDs have determinants that are super keys. Hence in BCNF.

GATE Question 9

Consider R(A,B,C,D,E) with FDs: AB → C, C → D, D → E. The relation is in which normal form?

✓Answer: 2NF but not 3NF

Explanation:

Candidate key is AB. Non-prime attributes: C, D, E. No partial dependency (C depends on full key AB). But there's transitive dependency: AB → C → D → E. Non-prime E depends on non-prime D through C. So in 2NF but not 3NF.

GATE Question 10

A relation is in 3NF but not in BCNF. Which of the following must be true?

✓Answer: There exists a FD X → Y where X is not a super key but Y is a prime attribute

Explanation:

For 3NF: Either X is super key OR Y is prime attribute. For BCNF: X must be super key for all non-trivial FDs. So if in 3NF but not BCNF, there must be FD where X is not super key, but Y is prime (allowed in 3NF, not in BCNF).

Chapter 23: GATE Questions - SQL

GATE Question 11

What is the output of: SELECT COUNT(*) FROM R WHERE A > 10 AND A < 5?

✓Answer: 0

Explanation:

No value can simultaneously be both greater than 10 AND less than 5. The WHERE condition is always false, so no rows match, COUNT returns 0.

GATE Question 12

Difference between TRUNCATE and DELETE:

✓Answer: TRUNCATE is DDL and faster, DELETE is DML and slower

Explanation:

TRUNCATE removes all rows instantly, resets identity, cannot be rolled back (in most DBMS), and is DDL. DELETE removes rows one by one, can have WHERE clause, can be rolled back, slower, and is DML.

GATE Question 13

In SQL, which join returns all rows from both tables?

✓Answer: FULL OUTER JOIN

Explanation:

FULL OUTER JOIN returns all rows from both tables. Where there's a match, it shows matched data. Where there's no match, it shows NULL for the missing side.

Chapter 24: GATE Questions - Transactions

GATE Question 14

Which of the following is NOT an ACID property?

✓Answer: All are ACID properties

Explanation:

ACID stands for: Atomicity (all-or-nothing), Consistency (valid state to valid state), Isolation (concurrent execution), Durability (permanent once committed). All four are essential ACID properties.

GATE Question 15

A schedule is conflict serializable if:

✓Answer: Its precedence graph is acyclic

Explanation:

To check conflict serializability: Create precedence graph with transactions as nodes, directed edge Ti → Tj for each conflicting operation pair where Ti comes before Tj. If graph has no cycles, schedule is conflict serializable.

GATE Question 16

Which schedule is recoverable? T1: R(A) W(A) Commit, T2: R(A) W(A) Commit where T2's R(A) reads value written by T1:

✓Answer: Only if T1 commits before T2

Explanation:

For schedule to be recoverable, if T2 reads value written by T1 (dirty read), then T1 must commit before T2. Otherwise, if T1 aborts after T2 commits, we cannot rollback T2, making it irrecoverable.

Chapter 25: GATE Questions - Concurrency Control

GATE Question 17

In 2PL protocol, a transaction:

✅Answer: Cannot acquire new locks after releasing any lock

Explanation:

2PL has two phases: Growing (acquire locks) and Shrinking (release locks). Once a transaction releases even one lock (enters shrinking phase), it cannot acquire any new locks. This ensures serializability but can cause deadlocks.

GATE Question 18

Timestamp ordering protocol is:

✅Answer: Deadlock-free but not cascadeless

Explanation:

Timestamp ordering prevents deadlock because transactions never wait for locks - they either proceed or abort. However, it allows dirty reads (reading uncommitted data), which can cause cascading rollbacks.

GATE Question 19

Strict 2PL protocol:

✅Answer: Holds all exclusive locks until commit

Explanation:

Strict 2PL requires holding all exclusive locks until transaction commits or aborts. This prevents cascading rollbacks and ensures recoverability. Shared locks can be released earlier in basic strict 2PL variant.

Chapter 26: GATE Questions - Indexing

GATE Question 20

In a B-tree of order m, maximum number of keys in a node is:

✅Answer: m - 1

Explanation:

B-tree of order m means maximum m children per node. Since number of keys = number of children - 1, maximum keys = m - 1. For example, order 4 B-tree has maximum 3 keys per node.

GATE Question 21

Minimum height of B-tree with n nodes and order m:

✅Answer: ⌈ log_m(n+1)⌉ - 1

Explanation:

Minimum height occurs when tree is maximally filled. Each node has maximum m children. Height h has at most m^(h+1) - 1 keys. Solving for h: h = ⌈ log_m(n+1)⌉ - 1.

GATE Question 22

In B+ tree, all data pointers are in:

✅Answer: Leaf nodes only

Explanation:

B+ tree stores all data pointers (record pointers) only in leaf nodes. Internal nodes contain only keys and tree pointers for navigation. Leaf nodes are linked for sequential access. This makes range queries very efficient.

QUICK REFERENCE TABLES

Normal Forms Summary

Normal Form	Requirements	Eliminates
1NF	Atomic values, No repeating groups	Multi-valued attributes
2NF	1NF + No partial dependency	Partial dependencies
3NF	2NF + No transitive dependency	Transitive dependencies
BCNF	3NF + Every determinant is super key	All anomalies except MVD
4NF	BCNF + No multi-valued dependency	Multi-valued dependencies
5NF	4NF + No join dependency	Join dependencies

Concurrency Control Protocols Comparison

Protocol	Deadlock-Free?	Cascadeless?	Serializability
Basic 2PL	No	No	Conflict
Strict 2PL	No	Yes	Conflict
Conservative 2PL	Yes	No	Conflict
Timestamp Ordering	Yes	No	Conflict
Strict TO	Yes	Yes	Conflict

SQL Join Types Summary

Join Type	Returns	NULL handling
INNER JOIN	Only matching rows	No NULLs
LEFT JOIN	All from left + matching from right	NULL for unmatched right
RIGHT JOIN	All from right + matching from left	NULL for unmatched left
FULL OUTER JOIN	All from both tables	NULL for unmatched both sides

CROSS JOIN	Cartesian product	No matching needed

CONCLUSION & EXAM TIPS

High-Weightage Topics for GATE:

22. Normalization (3NF, BCNF) - Practice decomposition
23. Functional Dependencies - Closure, candidate keys
24. Transactions & Concurrency - Serializability, 2PL
25. SQL Queries - JOINs, Nested queries, Aggregates
26. ER to Tables - Cardinality mapping
27. B-Trees & Indexing - Order, height calculations

Common Mistakes to Avoid:

- Confusing 3NF with BCNF - Check if determinant is super key
- Missing NULL handling in JOINs
- Forgetting GROUP BY with aggregate functions
- Incorrect closure calculation
- Not checking for cycles in precedence graphs

Final Tips:

□ IMPORTANT: Practice previous 10 years GATE questions. Focus on normalization and SQL - they appear every year!

Good luck with your preparation! □