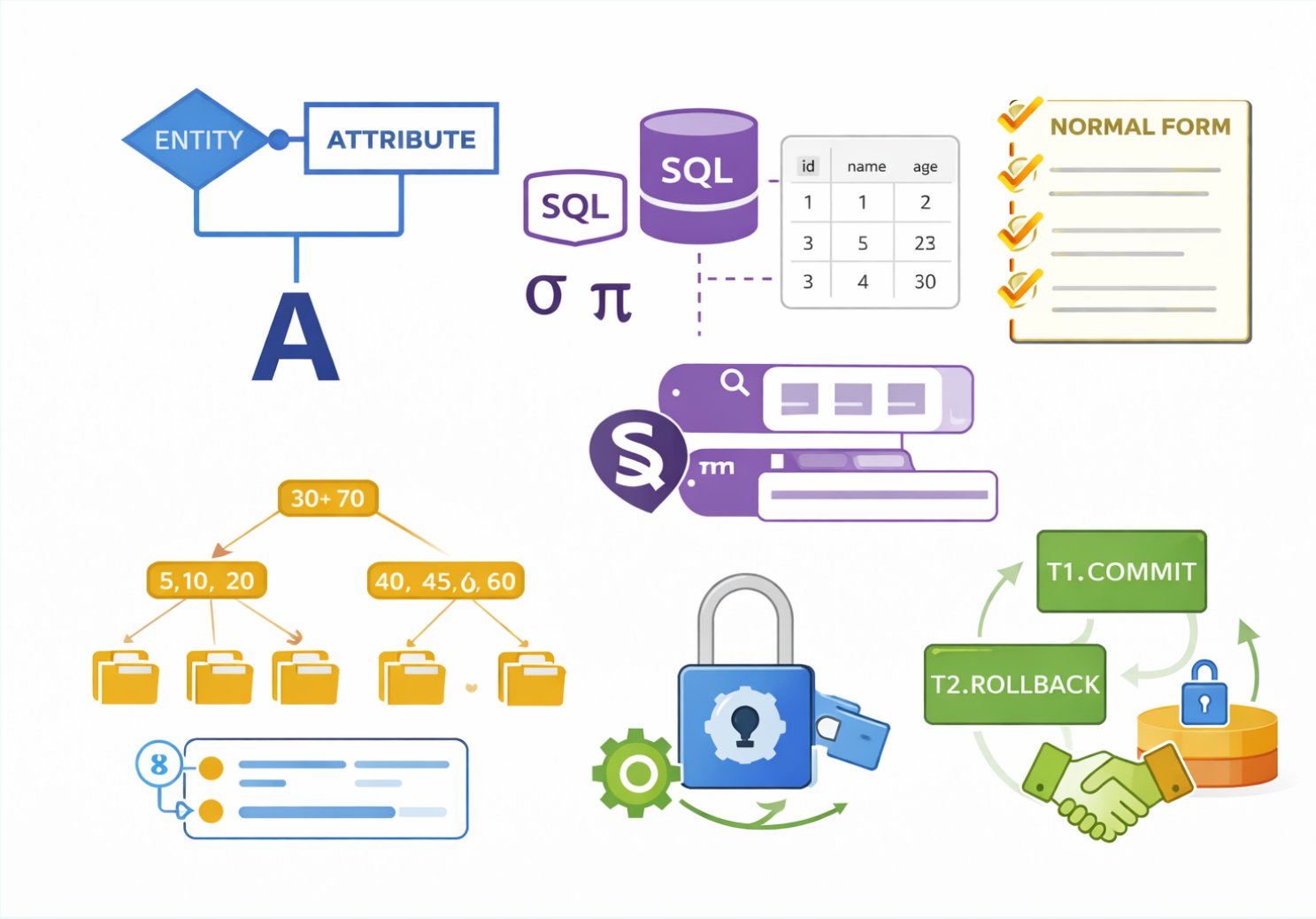
GATE – Computer Science Engineering (CSE)

Database Management Systems



GateXAIML

2026

Contents

About the Book

The **GATE Computer Science and Engineering (CSE) exam** serves as a national-level gateway to higher studies, research, and employment in top institutions and organizations. It evaluates a candidate's understanding of core Computer Science subjects, including Calculus, Matrices and Linear Algebra, Probability and Statistics, Discrete Mathematics, Algorithms, Data Structures, Theory of Computation, Databases, Operating Systems, and Computer Networks.

This book is **designed for aspirants of the GATE CSE exam**, focusing on **Database Management System**. It systematically covers theory, solved examples, and practice problems **aligned with the official syllabus**, helping learners build strong conceptual foundations and problem-solving skills.

Selected solutions and topic-wise lectures will be explained on my YouTube channel (@GATEXAiml), providing a **complete resource for GATE CSE preparation**.

Dedicated to all my Gurus and Students.
"Knowledge grows only when shared — and it must remain free, for that is how it thrives."

Database Management Systems- Syllabus

ER-model. Relational model: relational algebra, tuple calculus, SQL. Integrity constraints, normal forms. File organization, indexing (e.g., B and B+ trees). Transactions and concurrency control.

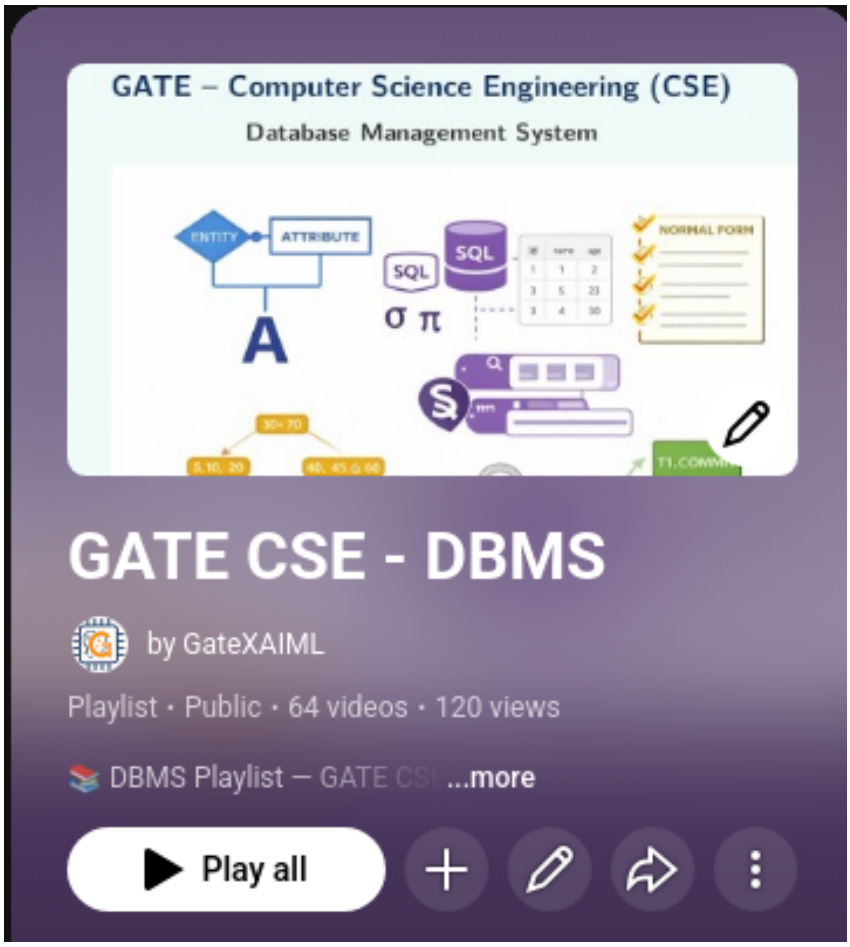
STOP!

Attention!

Some examples solved in video lectures are different from those given in this book.

The procedure to solve problems and examples is well explained in the video lectures, and it is highly recommended to go through the video lectures for complete understanding.

Official Video Playlist



Watch on YouTube

Chapter 1

Introduction

1.1 Overview

1.1.1 What is DBMS and Roles

DBMS

A **database-management system (DBMS)** is a collection of **interrelated data** along with a set of **programs** used to **access** those data. The collection of data, commonly called the **database**, stores **information relevant to an enterprise**. The primary objective of a **DBMS** is to provide a **convenient** and **efficient** way to **store** and **retrieve database information**.

- Database systems are designed to manage **large volumes of information**.
- Data management involves **defining storage structures** for information.
- It also includes providing **mechanisms for data manipulation**.
- A database system must ensure the **safety and reliability** of stored data.
- Data protection must be maintained even during **system crashes**.
- The system must prevent **unauthorized access** to information.
- When data are shared among multiple users, the system must avoid **anomalous and inconsistent results**.

1.1.2 Applications of DBMS

Applications

1. **Banking Systems**
   Databases store customer information, account details, transactions, loans, and credit records, ensuring data consistency, security, and concurrent access.
2. **Airline Reservation Systems**
   Database systems manage flight schedules, seat availability, passenger details, bookings, and cancellations while preventing double booking.
3. **University and Education Systems**
   Databases maintain student records, course registrations, grades, attendance, and faculty data, supporting multiple users.
4. **E-commerce Applications**
   Online platforms use databases to handle product catalogs, customer profiles, orders, payments, and inventory

management.

5. **Hospital and Healthcare Systems**
   Databases store patient records, medical histories, prescriptions, diagnostic reports, and billing information with strict access control.

6. **Telecommunication Systems**
   Database systems manage call detail records, customer plans, billing data, and network usage at very large scale.

7. **Social Media Platforms**
   Databases store user profiles, posts, comments, likes, messages, and relationships, enabling real-time data access.

8. **Library Management Systems**
   Databases track books, authors, issued and returned books, members, and fines efficiently.

9. **Government and E-Governance Systems**
   Databases manage citizen records, tax data, land records, voting information, and public service delivery systems.

10. **Manufacturing and Inventory Systems**
    Databases monitor production data, inventory levels, suppliers, orders, and supply chain operations.

1.1.3 Advantages Over File Processing Systems

Disadvantages of File-Processing Systems

Organizational information stored in a traditional file-processing system faces several major limitations.

Data Redundancy and Inconsistency

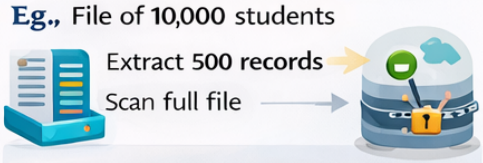
Same data stored in multiple files leads to storage wastage and inconsistency.

E.g., 1 student record= 2 KB
3 files
Storage= 6 KB
Edit 1 copy
Other copies not updated

Difficulty in Accessing Data

No ad-hoc querying leads to manual filtering or custom prog.

E.g., File of 10,000 students
Extract 500 records
Scan full file



E.g. Combining grades from one file & fee from another is complex.

Data Isolation

Data in various files and formats makes integration difficult.

E.g., File of 10,000 students
Extract 500 records

E.g. Disallow withdrawal needed if Account Balance < $0

Atomicity Problems

Partial updates due to failures can lead to data inconsistency.

E.g., Transfer $500 A → B

A = 3000 | B = 2000 | St
A = 2500 | B = 2000
$2500 – $2000
→ Disallow nedbed

A = 2500 → B = 2000 $280

Integrity Problems

Integrity checks scattered in programs make enforcement hard.

E.g., Account balance $2000
– Withdraw = $2500

A = 10,000 Withdrawals: $500
B = 10,000 Inconsistent state

Concurrent-Access Anomalies

Simultaneous operations by multiple users can give incorrect results.

E.g., A = 10,000; Withdrawals: $500 & $100 Concurrently

10,000 500 = 9500 vs. B. = 9900
Correct: 10,000 – 500 – 100 = $9400

Security Problems

No fine-grained access control limits data security.

E.g., Payroll needs salary data access and not academic

Security Problems

No fine-grained access control imits data security.

E.g., Payroll needs salary data access and not academic records

Advantages of DBMS

- **Data Redundancy and Inconsistency**
  In a file-processing environment, different application programs are developed independently over time. As a result, the same data may be stored in multiple files. For example, a student enrolled in both the Music and Mathematics departments may have personal details stored in two separate files. If the storage required for one student record is 2 KB and it is duplicated in 3 files, the total storage becomes 6 KB. If one copy is updated while others are not, the system enters an inconsistent state.

- **Difficulty in Accessing Data**
  File-processing systems do not support flexible queries. If a clerk needs a list of students living in a specific postal-code area, and no such program exists, the options are to manually filter data or write a new program. For example, extracting 500 relevant records from a file of $10{,}000$ students requires scanning the entire file, leading to inefficient data access.

- **Data Isolation**
  Data is scattered across multiple files with different formats, making integration difficult. For instance, combining student academic data stored in one file with fee details stored in another requires custom programs, increasing complexity and development time.

- **Integrity Problems**
  Integrity constraints must be enforced through application code. For example, consider a department account balance that must never fall below $\$0$. If an account has a balance of $\$2000$, a withdrawal of $\$2500$ should be disallowed. In file systems, such checks must be implemented in every relevant program, making updates and maintenance difficult.

- **Atomicity Problems**
  Operations must be executed completely or not at all. Consider a fund transfer of $\$500$ from account A to account B:

  $$A = 3000, \quad B = 2000$$

  After debit:

  $$A = 2500$$

  If a system failure occurs before crediting B, the database becomes inconsistent. Ensuring atomicity is difficult in file-processing systems.

- **Concurrent-Access Anomalies**
  Simultaneous data access by multiple users can lead to incorrect results. For example, if account A has a balance of $\$10{,}000$ and two withdrawals of $\$500$ and $\$100$ occur concurrently, both programs may read the same initial balance and write back $\$9500$ and $\$9900$. The correct balance should be:

  $$10{,}000 - 500 - 100 = 9400$$

- **Security Problems**
  File-processing systems lack fine-grained access control. For instance, payroll staff may need access only to salary data, not academic records. Since access control is embedded in individual programs, enforcing security policies consistently becomes difficult.

1.2 Data View

A **database system** consists of a collection of **interrelated data** along with a set of **programs** that enable users to **access** and **modify** this data. One of the primary objectives of a **database system** is to provide users with a **data view**—an **abstract representation** of the information—so that the system **hides the internal details** of how the data is **stored** and **managed**.

1.2.1 Data Models

Data Model

The foundation of a **database** is its **data model**, which provides a set of **conceptual tools** to describe the **data**, define the **relationships** among data elements, capture the **semantics** of the data, and specify **constraints** to maintain consistency.

Classification of Data Models

Data models can be broadly categorized into four types:

- **Relational Model**
  The **relational model** organizes data and the relationships among data using a collection of **tables** (also called **relations**). Each table consists of multiple **columns** (attributes), and each column has a unique name. The database is structured as **fixed-format records**, where each record corresponds to a row in the table. The relational model is a type of **record-based model** and is the most widely adopted model; the majority of modern database systems are based on it.
- **Entity-Relationship (E-R) Model**
  The **E-R model** represents data using **entities**—objects or things in the real world that are distinguishable from one another—and the **relationships** between these entities. This model is extensively used in **database design** to conceptually structure data.
- **Semi-Structured Data Model**
  The **semi-structured model** allows individual data items of the same type to have **different sets of attributes**, unlike relational or E-R models where all items of a type share the same attributes. Common formats include **JSON** and **XML**, which are widely used for representing semi-structured data.
- **Object-Based Data Model**
  With the rise of **object-oriented programming** (e.g., Java, C++, C#), databases evolved to support **object-oriented concepts**. In this model, **objects** can be stored in the database along with their **methods**, supporting **encapsulation** and **object identity**. Modern relational databases often integrate object-oriented features, effectively extending the relational model with objects and procedures stored in the database.

1.2.2 Data Abstraction

Data Abstraction

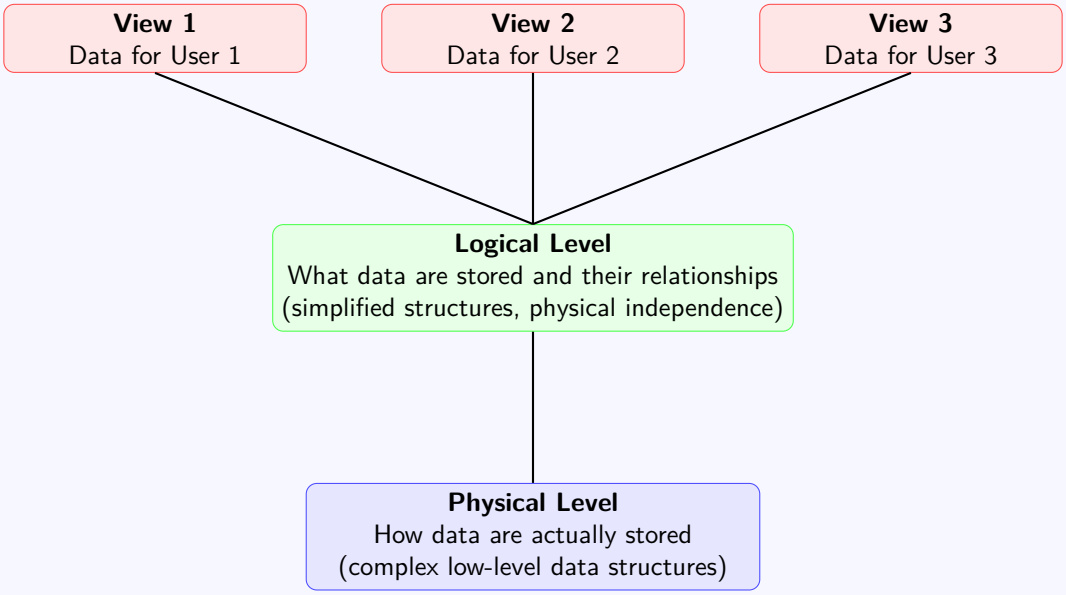
For a **database system** to be practical and effective, it must be able to **retrieve data efficiently**. To achieve this **efficiency**, developers employ **complex data structures** to represent data internally. However, since many **users** are not computer experts, the system **hides this complexity** using multiple **levels of data abstraction**, simplifying the way users interact with the database.

Levels of Data Abstraction

Database systems use multiple **levels of data abstraction** to simplify user interaction while hiding complexity. These levels are:

- **Physical Level or Internal**
  This is the **lowest level of abstraction** that describes **how data are physically stored**. It details the **complex low-level data structures** used to manage information efficiently.
- **Logical Level or Conceptual**
  The **middle level of abstraction** specifies **what data are stored** and the **relationships among them**. It uses **simplified structures** to represent the database while hiding the underlying physical complexity, providing **physical data independence**. **Database administrators** primarily interact with this level to determine which information is stored.

- **View Leve or Externall**
  The **highest level of abstraction** presents **only the portion of the database relevant to a user or application**. It simplifies user access and interaction by hiding unnecessary details. A database can provide **multiple views** tailored for different users' needs.



1.2.3 Data Independence

Data Independence

Data Independence is the ability of a database system to **modify the schema at one level** of abstraction without affecting the schema at the next higher level.
It ensures that changes in **storage details** or **logical structure** do not require rewriting application programs.
Data independence is a key advantage of a **DBMS** over traditional file systems.

Types of Data Independence

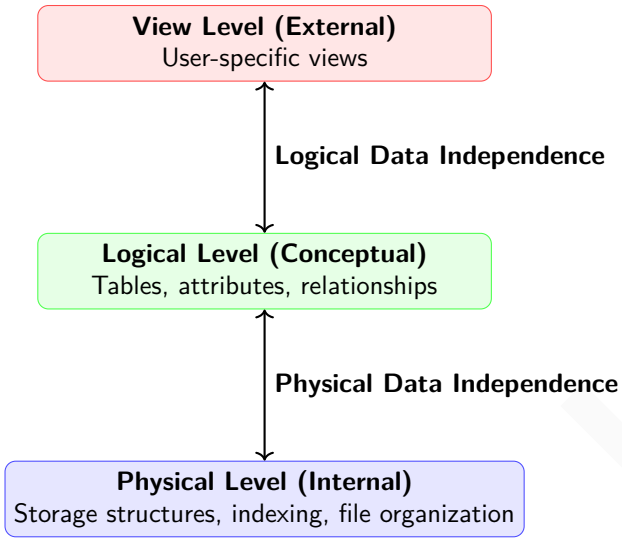
There are two types of data independence:

- **Physical Data Independence**
  The ability to change the **internal (physical) schema** without changing the **conceptual schema**. It hides changes in storage structure from users.

  Example: Changing file organization from *heap file* to *B+ tree index* or adding an index on `RollNo` does not affect the logical structure of the table.

- **Logical Data Independence**
  The ability to change the **conceptual schema** without changing the **external (view) schema**. It hides changes in logical structure from users.

  Example: Adding a new attribute `Email` to the `Student` table does not affect users who only access `Name` and `RollNo`.



Example 1: Example Illustrating Data Independence

Consider a database storing student information:
Conceptual Schema:

```
Student(RollNo, Name, Department, CGPA)
```

View for Faculty:

```
Student(RollNo, Name, CGPA)
```

View for Accounts Section:

```
Student(RollNo, Name)
```

Physical Change: Suppose the database administrator creates a **B+ tree index** on `RollNo`. This change affects only the **physical level**. Neither the conceptual schema nor the user views change. This is **Physical Data Independence**.
Logical Change: Suppose a new attribute `Email` is added:

```
Student(RollNo, Name, Department, CGPA, Email)
```

If faculty and accounts views remain unchanged, this demonstrates **Logical Data Independence**.

1.2.4 Database Schema and Instance

Schema and Instance

A **database** is dynamic, changing over time as **information is inserted** or **deleted**. The **collection of data** stored in the database at a particular moment is called an **instance** of the database. The **overall design** of the database is called the **database schema**.

This concept can be understood by analogy with a **program** in a programming language:

- The **database schema** corresponds to the **variable declarations** and their **types** in a program.
- The **instance of the database** corresponds to the **values of the variables** at a particular point in time.

Different Schema

Database systems can have multiple **schemas**, organized according to the **levels of abstraction**:

- **Physical Schema:** Describes the database design at the **physical level**.

- **Logical Schema:** Describes the database design at the **logical level**. This is the most important schema for **application programs**, as programmers construct applications using the logical schema.
- **View Schemas (Subschemas):** Describe **different views** of the database at the **view level**.

The **physical schema** is hidden beneath the logical schema and can usually be modified without affecting application programs. Application programs are said to have **physical data independence** if they do not depend on the physical schema, meaning they do not need to be rewritten when the physical schema changes.

1.2.5 Database Languages

Concept

A **database system** supports multiple query and control languages:

- A **Data-Definition Language (DDL)** is used to define the **structure and schema** of the database.
- A **Data-Manipulation Language (DML)** is used to **retrieve**, **insert**, **delete**, and **modify** data.
- A **Data-Control Language (DCL)** is used to **grant and revoke access permissions** on database objects.
- A **Transaction-Control Language (TCL)** is used to **manage transactions** and control changes using **commit, rollback, and savepoints**.

1.2.5.1 Data Definition Language (DDL)

DDL

A **Data-Definition Language (DDL)** is used to define the **structure of a database**, called the **database schema**.

- Using DDL, we specify:
  - tables (relations),
  - attributes and their data types,
  - relationships among tables,
  - constraints on data.
- DDL is also used to describe **storage structures** and **access methods** through a specialized form called **data storage and definition language**.
- These statements define **physical implementation details**, which are normally **hidden from users**.

Data Dictionary

- Execution of DDL statements produces **metadata**.
- Metadata is stored in the **data dictionary**.
- The data dictionary contains:
  - schema definitions,
  - constraints,
  - storage details,
  - authorization information.
- It is maintained **only by the database system**.
- The system consults it before accessing or modifying data.

1.2.5.2 Data Manipulation Language

DML

- A **Data-Manipulation Language (DML)** allows users to **access and modify data** stored in a database.
- DML operations are performed on data that are organized according to a **data model**.

Types of Data Access

- **Retrieval** of existing data from the database.
- **Insertion** of new data into the database.
- **Deletion** of existing data from the database.
- **Modification** (update) of stored data.

Categories of DML

- **Procedural DML**
  - User specifies **what data** are required.
  - User also specifies **how to retrieve** those data.
  - Emphasizes the **procedure or steps** of execution.
- **Declarative DML** (Nonprocedural DML)
  - User specifies only **what data** are needed.
  - User does **not specify how** to obtain the data.
  - The database system decides the **execution strategy**.

Comparison

- Declarative DMLs are generally **easier to learn and use**.
- They reduce the burden on users by hiding retrieval details.
- The database system must determine an **efficient access plan**.

1.2.5.3 Data Control Language

DCL

- A **Data Control Language (DCL)** is used to **control access and permissions** on database objects.
- DCL commands define **who can access** which data and **what operations** they are allowed to perform.
- It is mainly related to **security and authorization**.

Types of Access Control Operations

- **Granting** privileges to users or roles.
- **Revoking** privileges from users or roles.

Common DCL Commands with Examples

- **GRANT** — gives privileges

```
GRANT SELECT, INSERT ON student TO user1;
```

- **REVOKE** — removes privileges

```
REVOKE INSERT ON student FROM user1;
```

1.2.5.4 Transaction Control Language

TCL

- A **Transaction Control Language (TCL)** manages **transactions** in a database.
- TCL ensures **consistency and integrity** of data by controlling transaction execution.
- It works closely with the **ACID properties** of transactions.

Transaction Control Operations

- **Commit** — permanently saves changes.
- **Rollback** — undoes changes since last commit.
- **Savepoint** — creates intermediate rollback points.

Common TCL Commands with Examples

- **COMMIT** — make changes permanent

```
UPDATE account SET balance = balance - 500 WHERE acc_no = 101;
COMMIT;
```

- **ROLLBACK** — undo uncommitted changes

```
DELETE FROM orders WHERE order_id = 9001;
ROLLBACK;
```

- **SAVEPOINT** — set a checkpoint

```
SAVEPOINT sp1;
UPDATE employee SET salary = salary * 1.1;
ROLLBACK TO sp1;
```

1.3 Database Engine and Core Internal Components

1.3.1 Database Engine — Overview

Database Engine

- The **database engine** is the core execution layer of a DBMS.
- It is responsible for:
  - executing queries,
  - managing stored data,

  - enforcing rules and constraints,
  - handling transactions safely.
- It acts as the bridge between:
  - user queries and application programs, and
  - the actual data stored on disk.
- Internally, the database engine mainly consists of:
  - **Storage Manager**
  - **Query Processor**
  - **Transaction Manager**

1.3.2 Storage Manager

Storage Manager

- The **storage manager** connects high-level database operations with low-level physical storage.
- It serves as the interface between:
  - application programs and queries, and
  - the operating system file system.
- Raw data are stored on disk using OS file services.
- The storage manager converts **DML commands** into **low-level file operations**.
- It is responsible for:
  - storing data,
  - retrieving data,
  - updating data efficiently.

Main Components of Storage Manager

- **Authorization and Integrity Manager**
  - Verifies user access rights.
  - Enforces integrity constraints.
  - Prevents unauthorized data access or invalid updates.
- **Transaction Manager**
  - Ensures database correctness during failures.
  - Coordinates concurrent operations safely.
  - Prevents conflicting updates.
- **File Manager**
  - Controls disk space allocation.
  - Maintains physical file structures.
  - Organizes how records are stored on disk.
- **Buffer Manager**

  - Moves data between disk and main memory.
  - Caches frequently used data in RAM.
  - Decides which pages remain in memory and which are replaced.
  - Enables databases larger than RAM to be processed efficiently.

Data Structures Managed by Storage Layer

- **Data Files**
  - Store the actual database records.
- **Data Dictionary (Metadata Repository)**
  - Stores schema definitions.
  - Contains structure and constraint information.
  - Used internally by the DBMS.
- **Indexes**
  - Provide fast lookup of records.
  - Store pointers to matching data rows.
  - Example: index on EmployeeID speeds up search.

1.3.3 Query Processor

Query Processor

- The **query processor** handles query interpretation and execution.
- It converts user queries into efficient execution steps.

Query Processor Components

- **DDL Interpreter**
  - Processes schema definition commands.
  - Updates metadata in the data dictionary.
- **DML Compiler**
  - Translates queries into low-level execution plans.
  - Generates multiple alternative plans.
  - Performs **query optimization**.
  - Chooses the lowest-cost execution strategy.
- **Query Evaluation Engine**
  - Executes the selected execution plan.
  - Runs low-level database operations.

1.3.4 Transaction Management

Transaction Management

- A **transaction** is a set of database operations forming one logical task.
- Example: transfer money from Account A to Account B.

Transaction Execution Rules

- Each transaction must preserve database correctness.
- Temporary inconsistency may occur during execution.
- Final committed state must be consistent.

Essential Transaction Properties (ACID)

- **Atomicity**
  - A transaction executes completely or not at all.
  - If any step fails, all prior changes are rolled back.
  - No partial updates are allowed.
- **Consistency**
  - A transaction must preserve all integrity constraints.
  - Database moves only from one valid state to another valid state.
  - Rules, keys, and constraints must remain satisfied after commit.
- **Isolation**
  - Concurrent transactions must not interfere with each other.
  - Intermediate results of one transaction are hidden from others.
  - Effect is equivalent to serial (one-by-one) execution.
- **Durability**
  - Once a transaction is committed, its results are permanent.
  - Changes survive system crashes and power failures.
  - Achieved using logs and recovery mechanisms.

Transaction Manager Structure

- Transaction manager consists of:
  - **Concurrency-control manager**
  - **Recovery manager**

Failure Handling

- If a transaction fails:
  - its effects must be completely undone.
  - database must be restored to the earlier state.
- This is handled by the **recovery manager**.

Concurrency Control

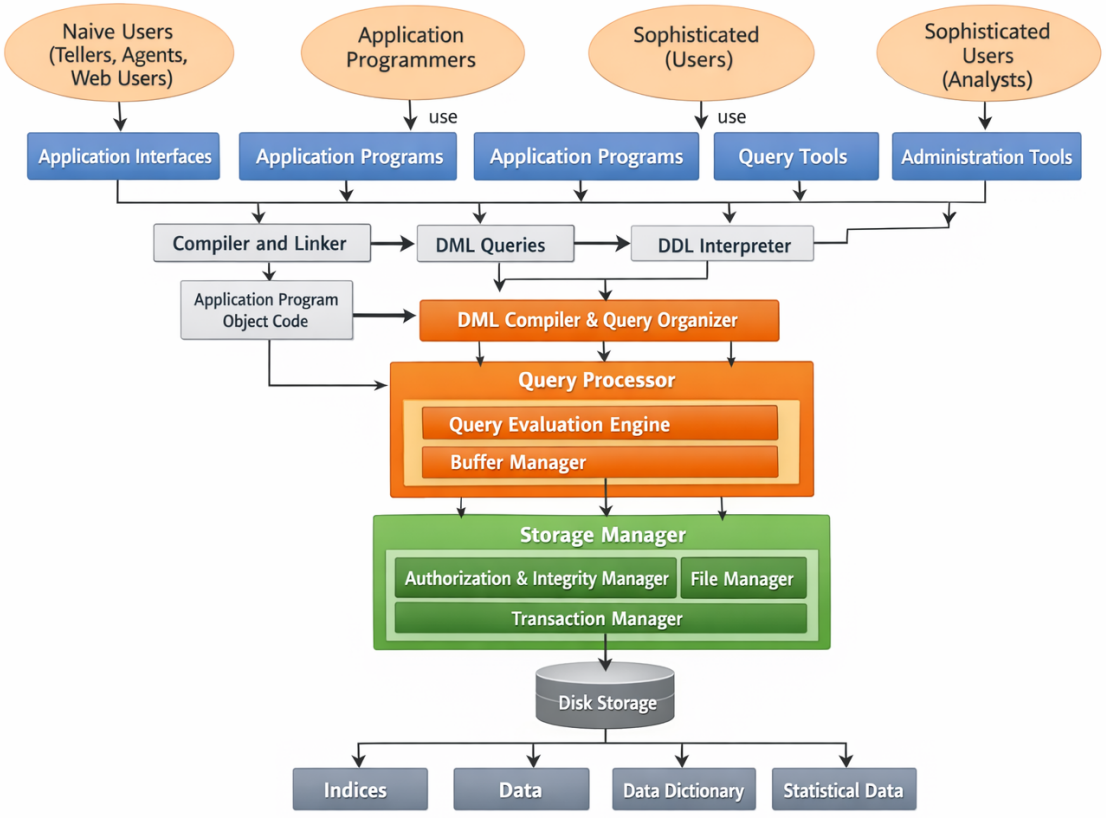
- Multiple transactions may run simultaneously.
- Concurrent updates can cause conflicts.
- The **concurrency-control manager**:
  - coordinates concurrent execution,
  - prevents data inconsistency,
  - enforces isolation.

1.3.5 Database and Application Architecture

The figure shows the internal structure of a Database Management Systems(DBMS) and how different types of users interact with database components through multiple processing layers.

Types of Users

- **Naive Users** — use predefined interfaces (forms, web apps, teller screens).
- **Application Programmers** — write application programs using DML calls.
- **Sophisticated Users (Analysts)** — directly use query tools for analysis.
- **Database Administrators (DBA)** — use administration tools for control and maintenance.



Database Architecture

User Access Paths

- Naive users → Application Interfaces
- Programmers → Application Programs
- Analysts → Query Tools
- DBA → Administration Tools

Compilation Layer

- Application programs are processed by the **Compiler and Linker**.
- This produces **Object Code**.
- DML queries inside programs are extracted for further processing.

Query Processing Layer

- **DDL Interpreter** — processes schema definitions and updates metadata.
- **DML Compiler and Organizer** — parses and optimizes DML queries.
- Output is given to the **Query Evaluation Engine**.

Query Processor Components

- **Query Evaluation Engine** — executes optimized queries.
- **Buffer Manager** — manages data pages in main memory.

Storage Manager Layer

- **Authorization & Integrity Manager** — checks permissions and constraints.
- **Transaction Manager** — ensures ACID properties.
- **File Manager** — handles file organization on disk.

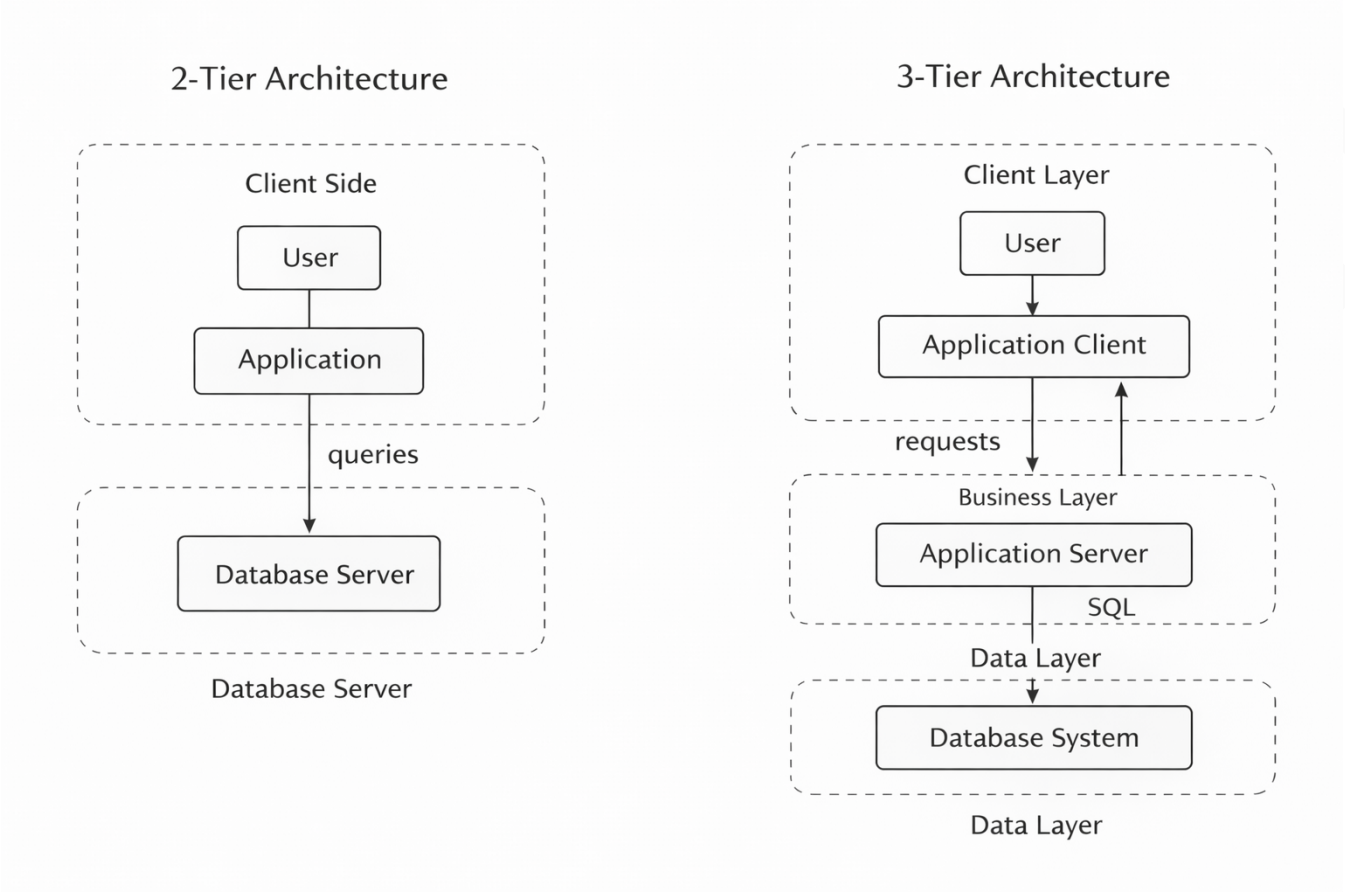
Disk Storage Contents

- Actual **Data Tables**
- **Indexes** for fast access
- **Data Dictionary** (metadata)
- **Statistical Data** for query optimization

End-to-End Workflow

- User submits request (query/program/interface).
- Query or program is compiled or interpreted.
- DML is optimized by the query processor.
- Storage manager checks authorization and integrity.
- Buffer manager fetches required data pages.
- Data is read/written on disk.
- Results are returned to the user.

1.3.5.1 Two-Tier vs Three-Tier Database Application Architecture



Two-Tier Architecture

- In two-tier architecture, the application runs directly on the client machine.
- The client program sends database queries directly to the database server.
- The database server processes queries and returns results to the client.
- The client contains:
  - User interface
  - Application logic
  - Database query calls (SQL/DML)
- Communication happens directly between:

$$\text{Client Application} \leftrightarrow \text{Database Server}$$

- Characteristics:
  - Simple design
  - Suitable for small systems
  - Harder to maintain when many clients exist
  - Lower security (database exposed to clients)
  - Business logic duplicated across clients

Three-Tier Architecture

- In three-tier architecture, the client is only a front end.
- The client does **not** directly talk to the database.
- A separate **application server** sits between client and database.
- All business logic is placed in the application server.
- Communication flow:

$$\text{Client} \leftrightarrow \text{Application Server} \leftrightarrow \text{Database Server}$$

- Layer roles:
  - **Client Layer:**
    - Web browser or mobile app
    - Handles UI only
  - **Application Server Layer:**
    - Contains business rules
    - Validates requests
    - Builds database queries
    - Controls transactions
  - **Database Layer:**
    - Stores and retrieves data
    - Executes queries
- Advantages:
  - Better security (database not directly exposed)
  - Better scalability
  - Easier maintenance
  - Centralized business logic
  - Improved performance under heavy load

Workflow Comparison

- Two-tier:
  - User $\rightarrow$ Client App $\rightarrow$ Database $\rightarrow$ Client App $\rightarrow$ User
- Three-tier:
  - User $\rightarrow$ Client $\rightarrow$ Application Server $\rightarrow$ Database $\rightarrow$ Application Server $\rightarrow$ Client $\rightarrow$ User

1.4 Problems

Problem 1 (MSQ) *Which of the following are distinct advantages of a Database Management Systems(DBMS) over a traditional File Processing System?*
A. Reduction in data redundancy and inconsistency by centralizing data storage.
B. Support for atomic transactions, ensuring that either all operations or none are performed.
C. Elimination of the need for primary keys and unique identifiers to locate data.
D. Provision of data isolation, allowing multiple applications to access private data without interference.

Problem 2 *Consider the three-level architecture. At which level is the **Physical Data Independence** achieved?*
A. Between the External level and the Conceptual level.
B. Between the Conceptual level and the Internal level.
C. Within the External level only.
D. Between the User View and the Logical schema.

Problem 3 (MSQ) *Which of the following statements regarding Database Schemas and Instances are correct?*
A. The database schema is the logical design, which is expected to change very infrequently.
B. A database instance is the actual content of the database at a specific point in time.
C. The schema is analogous to variable declarations in a programming language, whereas the instance is analogous to the values of those variables.
D. Schema evolution is a common daily task performed during the data manipulation phase.

Problem 4 *A Data Definition Language (DDL) is used primarily to define the database schema. When a DDL statement is executed, the output is typically stored in a special file called:*
A. The Query Execution Plan.
B. The Data Dictionary (or Metadata Repository).
C. The Transaction Log.
D. The Buffer Pool.

Problem 5 (MSQ) *Regarding the **Storage Manager** component of a database engine, which of the following tasks are its responsibility?*
A. Mapping DDL statements into a set of metadata tables.
B. Interaction with the File Manager to store raw data on the disk.
C. Maintaining consistency between data in the buffer and data on the physical disk.
D. Ensuring data remains consistent despite system crashes via the recovery manager.

Problem 6 *In the context of the **Query Processor**, the "Query Optimizer" is responsible for:*
A. Translating SQL into C++ code for execution.
B. Selecting the lowest-cost evaluation plan from multiple logically equivalent strategies.
C. Ensuring that the user has the required permissions to access the table.
D. Locking the rows to prevent other users from modifying them during the query.

Problem 7 (MSQ) *The **Transaction Manager** ensures the ACID properties. Which of the following scenarios describes a violation of "Atomicity"?*
A. A fund transfer debiting Account A but failing to credit Account B due to a crash.
B. Two users booking the same single seat on a flight simultaneously.
C. A database losing a successfully committed transaction after a power failure.
D. A user seeing an intermediate, uncommitted sum while another transaction is still calculating it.

Problem 8 *In a **Three-Tier Architecture**, the "Business Logic" typically resides in which layer?*
A. The Database Server tier.
B. The Client (User Interface) tier.

C. The Application Server (Middleware) tier.
D. The Storage Manager tier.

Problem 9 *Logical Data Independence is generally considered harder to achieve than Physical Data Independence because:*
A. Disk hardware changes more frequently than business rules.
B. It requires changing the mapping between the external view and the conceptual schema when the conceptual schema changes.
C. The internal level is invisible to the Database Administrator.
D. DDL compilers cannot process conceptual changes automatically.

Problem 10 *Which of the following commands is typically classified as a **Data Manipulation Language (DML)** command in its non-procedural form?*
A. ALTER TABLE
B. GRANT CONNECT
C. SELECT
D. CREATE INDEX

Problem 11 *Which of the following commands is primarily associated with **Data Control Language (DCL)**?*
A. COMMIT
B. GRANT
C. INSERT
D. CREATE

Problem 12 *The SQL command used to **remove previously assigned privileges** from a user is:*
A. DENY
B. ROLLBACK
C. REVOKE
D. DROP

Problem 13 *Which of the following commands belongs to **Transaction Control Language (TCL)**?*
A. SAVEPOINT
B. UPDATE
C. ALTER
D. SELECT

Problem 14 *After executing several UPDATE statements in a transaction, which command makes the changes **permanent** in the database?*
A. GRANT
B. COMMIT
C. REVOKE
D. TRUNCATE

Problem 15 (MSQ) *Which of the following components are part of the **Database Engine's Internal Query Processor**?*
A. DDL Interpreter.
B. DML Compiler.
C. Buffer Manager.
D. Query Evaluation Engine.

Problem 16 *A "Physical Data Model" is best described as a model that:*
A. Describes how data is stored as records and files on the disk, including indices and pointers.
B. Describes the relationship between entities using E-R diagrams.
C. Represents the user's view of the data without any implementation details.
D. Focuses on the logical constraints like Primary Keys and Foreign Keys.

Problem 17 (MSQ) *Data Abstraction is used to hide complexity from users. Which statements are correct regarding the levels of abstraction?*
A. The View Level is the highest level, hiding details of the entire database structure.
B. The Logical Level describes what data are stored and what relationships exist among those data.
C. The Physical Level is the lowest level, describing how the data are actually stored.
D. Database Administrators (DBAs) work exclusively at the View Level.

Problem 18 *What is the primary role of the **Buffer Manager** within the Storage Manager?*
A. To compile SQL queries into machine-level instructions.
B. To decide which data should be cached in main memory to minimize disk I/O.
C. To manage the passwords and security of users.
D. To ensure that transactions follow the Serializability property.

Problem 19 *The "Declarative" nature of DML (like SQL) implies that:*
A. The user must specify the exact algorithm to find the data.
B. The user specifies what data is needed without specifying how to get it.
C. The database engine does not need to optimize the query.
D. Only one record can be retrieved at a time.

Problem 20 *Which component is responsible for ensuring the "Durability" of a transaction?*
A. Authorization Manager.
B. Query Optimizer.
C. Recovery Manager.
D. DDL Interpreter.

Problem 21 (MSQ) *Which of the following are examples of **Metadata** stored in the Data Dictionary?*
A. Names of the tables and their attributes.
B. Integrity constraints (e.g., NOT NULL, Primary Key).
C. The actual salary of an employee named "John".
D. Information about indices created on the tables.

Problem 22 *A "Procedural DML" differs from a "Declarative DML" because it requires the user to:*
A. Define the schema of the data.
B. Specify how to perform the data retrieval using control structures.
C. Use only graphical interfaces.
D. Ignore the logical level of abstraction.

Problem 23 *Which of the following best describes the "Consistency" property of a transaction?*
A. The database must be in a valid state before and after the transaction.
B. The data must be stored on multiple disks for safety.
C. Multiple transactions must run one after another.
D. The transaction must complete in a fixed amount of time.

Problem 24 *In the database system architecture, the "Authorization and Integrity Manager" is a sub-component of:*
A. The Query Processor.
B. The Storage Manager.
C. The Network Interface.
D. The Transaction Manager.

Problem 25 (MSQ) *Which of the following are Data Models?*
A. Relational Model.

B. Object-Oriented Model.
C. Entity-Relationship Model.
D. Storage Management Model.

Problem 26 *In a university database, if the Registrar adds a new attribute "Email" to the "Student" table, but the existing application that generates Grade Reports continues to work without any modification, this is an example of:*
A. Physical Data Independence.
B. Logical Data Independence.
C. Data Redundancy.
D. Transaction Isolation.

Problem 27 *In the three-level database architecture, the **Mapping** between the levels is crucial. If the conceptual schema is changed by adding a new attribute to a table, which of the following is necessary to maintain **Logical Data Independence** for an existing user view?*
A. The physical storage structures must be reorganized.
B. The mapping between the external level and the conceptual level must be updated.
C. The entire database instance must be deleted and re-imported.
D. All existing DML queries in the application must be rewritten to include the new attribute.

Problem 28 (MSQ) *Which of the following statements correctly characterize **Physical Data Independence**?*
A. It allows the DBA to change the storage structures (e.g., switching from B-trees to Hashing) without changing the conceptual schema.
B. It ensures that the application programs do not need to be modified when the file organization on the disk changes.
C. It is generally more difficult to achieve than Logical Data Independence.
D. It is provided by the separation between the Conceptual level and the Internal (Physical) level.

Problem 29 (MSQ) *The **Conceptual Level** (or Logical Level) of abstraction is often described as the "DBA's view" of the database. Which of the following tasks are performed at this level?*
A. Defining the data types of the attributes.
B. Defining the security constraints and semantic integrity rules.
C. Specifying the record offsets and buffer page sizes.
D. Identifying the entities and relationships for the entire organization.

Problem 30 (MSQ) *Which of the following would require a change in the **External Schema** while keeping the Conceptual Schema intact?*
A. A specific department needs to hide the "Salary" column from its junior clerks.
B. The database administrator decides to partition a large table into two physical files.
C. A new user group joins the organization and requires a specific subset of data formatted differently.
D. The primary key of the Employee table is changed from SSN to EmployeeID.

Problem 31 *The term **"Schema"** refers to the stable framework of the database. If we say that "The number of students currently enrolled is 500," we are describing:*
A. The Logical Schema.
B. The Physical Schema.
C. The Database Instance.
D. The View Metadata.

Problem 32 *In the context of data independence, the "Transparency" of physical storage details to the logical user is primarily the responsibility of:*
A. The Operating System's File Manager.
B. The DBMS Database Engine (Mapping processes).
C. The Application Programmer.

D. The End User.

Problem 33 *What is the primary disadvantage of maintaining a high degree of* ***Data Independence*** *in a DBMS?*
A. Increased data redundancy.
B. Higher overhead in query processing due to the need for level-to-level mapping.
C. Reduced security.
D. Inability to use SQL as a declarative language.

1.5 Try it Yourself

Exercise 1 *Which of the following best explains why a DBMS is preferred over a file-processing system for a banking application?*
A. It allows data to be stored in multiple separate files to improve access speed.
B. It provides a mechanism to ensure that multiple concurrent updates do not lead to inconsistent data.
C. It allows the programmer to define the exact physical offset of records on the disk.
D. It eliminates the need for any secondary storage like Hard Disk Drives.

Exercise 2 *In the context of data abstraction, the "Logical Level" is responsible for describing:*
A. The specific bit-level representation of data on the storage media.
B. What data is stored in the database and what relationships exist among that data.
C. The specific subset of the database that is visible to a non-technical end user.
D. The hardware-specific instructions for the disk read/write head.

Exercise 3 *Physical Data Independence is defined as the ability to modify the _____ without causing _____ to be rewritten.*
A. Conceptual schema; application programs
B. Physical schema; conceptual schema
C. Application programs; physical schema
D. View level; logical level

Exercise 4 (MSQ) *Which of the following are responsibilities of the **Storage Manager** in a database engine?*
A. Managing the space allocation on the disk.
B. Ensuring that the database remains in a consistent state after a system crash.
C. Optimizing the execution path of an SQL query.
D. Buffering data in main memory to reduce disk latency.

Exercise 5 *A "Database Schema" is to a "Database Instance" as:*
A. A variable's value is to the variable's data type.
B. A programming language's compiler is to its interpreter.
C. A class definition is to an object in memory.
D. A hard drive is to the data stored on it.

Exercise 6 *The "Atomicity" property of a transaction ensures that:*
A. Only one user can access a record at any given time.
B. Data is encrypted before being stored on the disk.
C. All operations of a transaction are completed, or none are performed at all.
D. Once a transaction is committed, its changes survive a power failure.

Exercise 7 *Which of the following is an example of a **Data Definition Language (DDL)** operation?*
A. Modifying the salary of all employees by 10%.
B. Adding a new integrity constraint that prevents negative values in a 'Quantity' column.
C. Searching for the name of the student with the highest GPA.
D. Deleting the records of students who have graduated.

Exercise 8 *In a database engine, the component that translates a user's declarative request into a sequence of low-level instructions is the:*
A. Transaction Manager
B. Query Processor
C. Buffer Manager
D. DDL Interpreter

Exercise 9 (MSQ) *Which of the following scenarios demonstrate a failure of **Data Isolation** in a file-processing system?*
A. Two different programs are unable to share the same data file due to different file formats.
B. A user updates a file while another user is reading it, leading to the reader seeing partial data.
C. A program crashes and leaves a file in a corrupted state.
D. Multiple applications store redundant copies of the same data.

Exercise 10 *Logical Data Independence is violated if:*
A. Moving a database to a new server requires updating the IP address in the application.
B. Splitting one table into two for normalization requires changing the queries in the application programs.
C. Upgrading from an HDD to an SSD requires changing the SQL syntax.
D. The user is able to see the B-Tree structure used for indexing.

Exercise 11 *The **Data Dictionary** (Metadata) typically contains all of the following EXCEPT:*
A. The names and types of columns in a table.
B. The username of the account that created a table.
C. The actual rows of data stored inside a table.
D. The definitions of views and indexes.

Exercise 12 *What is the primary goal of the **Query Optimizer**?*
A. To check the syntax of the SQL statement.
B. To find the most efficient way to execute a query by comparing different strategies.
C. To verify that the user has the 'Read' permission for the requested table.
D. To convert the results of a query into a graphical format.

Exercise 13 (MSQ) *Which properties constitute the **ACID** requirements for transaction management?*
A. Consistency
B. Availability
C. Isolation
D. Durability

Exercise 14 *A "Procedural DML" is one in which the user:*
A. Only describes what data is needed.
B. Specifies both what data is needed and how to retrieve that data.
C. Defines the structure of the database tables.
D. Grants permissions to other users.

Exercise 15 *Which component of the database engine is responsible for maintaining the "Durability" of transactions?*
A. Query Evaluation Engine
B. Authorization Manager
C. Recovery Manager
D. DDL Compiler

Exercise 16 *Which command is used to **undo all changes** made in the current transaction since the last commit?*
A. ROLLBACK
B. REVOKE
C. SAVEPOINT
D. GRANT

Exercise 17 *A database administrator creates an intermediate checkpoint inside a transaction so that partial undo is possible later. Which TCL command is used?*
A. CHECKPOINT
B. SAVEPOINT
C. COMMIT
D. FLASHBACK

Exercise 18 *Which of the following statements is true about **DCL commands**?*
A. They modify table structure.
B. They control user privileges and access rights.
C. They are used only for transaction rollback.
D. They retrieve tuples from relations.

Exercise 19 *The separation of the "Application Logic" from the "Data Storage" in a multi-tier system is primarily intended to improve:*
A. Disk read/write speeds.
B. Scalability and security of the system.
C. The resolution of the user interface.
D. The capacity of the physical hard drives.

Exercise 20 *If a system allows "Concurrency Control", it means:*
A. Data is stored on multiple servers at the same time.
B. Multiple users can modify the same data simultaneously without causing inconsistency.
C. The database can perform backups while users are logged in.
D. Every transaction is automatically mirrored on a secondary disk.

Exercise 21 *A user view (External Level) is used to:*
A. Improve the physical storage efficiency of the database.
B. Provide a simplified or restricted version of the database to a specific group of users.
C. Directly manage the RAM allocation for the database engine.
D. Define the primary keys for the entire organization.

Exercise 22 (MSQ) *Which of the following are examples of **Metadata**?*
A. The maximum length allowed for a 'Username' field.
B. The date a specific record was last modified.
C. The fact that the 'Total' column is calculated by adding 'Price' and 'Tax'.
D. The list of all indexes currently active on the 'Orders' table.

Exercise 23 *The "Physical Level" of data abstraction is concerned with:*
A. The logical relationships between entities.
B. How the data is actually stored in terms of blocks, pointers, and files.
C. The names of the tables in the database.
D. The business rules that govern the data.

Exercise 24 *When a DDL statement is compiled, the resulting output is usually placed in:*
A. The transaction log.
B. The system catalog or data dictionary.
C. The query cache.
D. The application source code.

Exercise 25 *"Data Inconsistency" occurs when:*
A. Two records have different values for the same primary key.
B. Redundant copies of the same data are not updated simultaneously.
C. A user forgets their password.
D. The database is stored on an unformatted disk.

Exercise 26 *The **Transaction Manager** is primarily responsible for:*
A. Writing SQL queries for the user.
B. Ensuring that the database stays consistent despite system failures and concurrent access.
C. Designing the E-R diagram for the database.
D. Compiling DDL statements into a data dictionary.

Exercise 27 *Which of the following is **not** a task performed by the Query Processor?*
A. Syntax checking of SQL commands.
B. Semantic analysis of table names.
C. Managing the lock table for concurrent transactions.
D. Plan generation for join operations.

Exercise 28 (MSQ) *Which statements regarding the "Declarative" nature of SQL are true?*
A. It allows users to focus on what they want rather than how to get it.
B. It makes the code more portable across different database engines.
C. It eliminates the need for a Query Optimizer.
D. It is generally easier for end-users than procedural languages.

Exercise 29 *"Data Atomicity" in a DBMS is specifically designed to handle:*
A. Security breaches by unauthorized users.
B. Partial failures of a system during an update process.
C. Slow network connections.
D. Redundant data entries.

Exercise 30 *The **Admission Control** component of a database engine determines:*
A. Which users are allowed to create new tables.
B. How many transactions can be processed concurrently to avoid system overload.
C. Which attributes are allowed to be NULL.
D. Which physical disk block should be used for a new record.

Exercise 31 *In a database system, the "Buffer Manager" interacts most directly with:*
A. The Graphical User Interface (GUI).
B. The Disk File Manager.
C. The DDL Interpreter.
D. The SQL Formatter.

Exercise 32 *"Durability" in transaction management ensures that:*
A. Transactions are executed very quickly.
B. Once a transaction is finished, its results are not lost even in a system crash.
C. The data is always consistent.
D. No two users can see the same data at once.

Exercise 33 *Which layer of abstraction is closest to the end user?*
A. Conceptual Level
B. Internal Level
C. View Level
D. Physical Level

1.6 YouTube Links and QR Codes

Lecture	Details	YouTube Link	QR Code
1	Introduction & Advantages over File System	https://youtu.be/7Rwu0npZSTc	
2	Data Models — Data Independence — Levels — Database Languages (DDL, DML, DCL, TCL)	https://youtu.be/4cSqjBsGpwc	
3	Database Architecture — DB Engine & Application Architecture Explained	https://youtu.be/n9LFSeczES4	
4	Problem Solving on DB Languages, Data Abstraction & DB Architecture	https://youtu.be/fR2goPCBLmQ	

Chapter 2

ER Model

2.1 Entity–Relationship (E–R) Data Model

Concept

The Entity–Relationship (E–R) data model was introduced to simplify database design by providing a structured method to define an enterprise schema. This schema represents the overall logical structure of a database.
The E–R model helps convert real-world objects, their properties, and their interactions into a conceptual database design. Due to its clarity and effectiveness, many database design tools are based on E–R modeling concepts.

Primary Components

The model is built on three primary components:

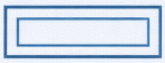
- Entity sets
- Relationship sets
- Attributes

The conceptual design created using this model is typically represented using an **E–R diagram**, which gives a graphical view of entities, relationships, and their attributes.

2.1.1 Symbols used in ER Diagrams

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

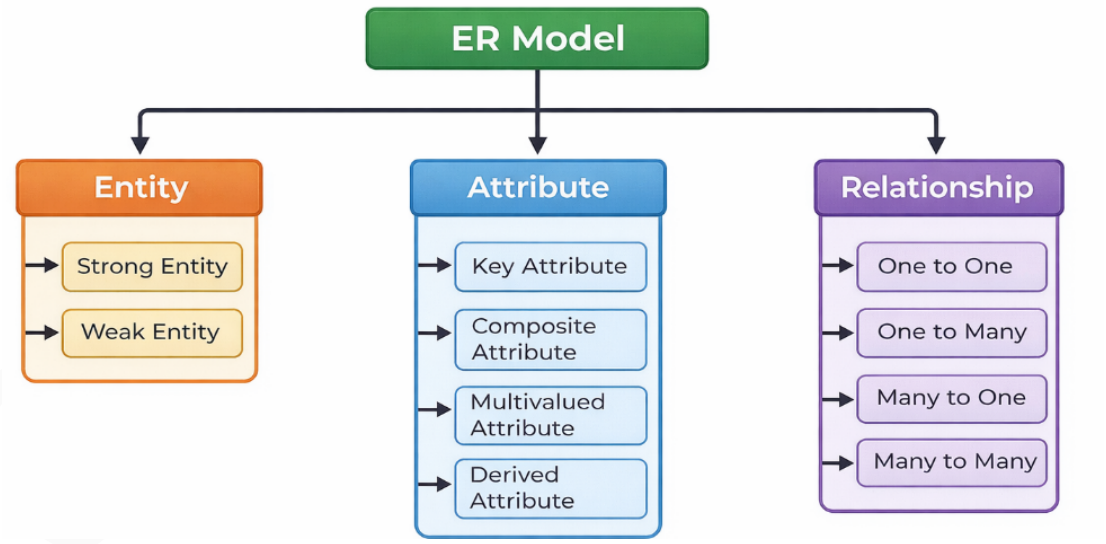
- **Rectangles:** Rectangles represent Entities in the ER Model.
- **Ellipses:** Ellipses represent Attributes in the ER Model.
- **Diamond:** Diamonds represent Relationships among Entities.
- **Lines:** Lines represent attributes to entities and entity sets with other relationship types.
  - **Single Lines:** Single lines represent the partial participation from the entity in an relationship.
  - **Double Lines:** Double lines represent the total participation from the entity in an relationship.
- **Double Ellipse:** Double Ellipses represent Multi-Valued Attributes.
- **Double Rectangle:** Double Rectangle represents a Weak Entity.

Figures	Symbols	Represents
Rectangle		Strong Entity
Ellipse		Attribute
Diamond		Relationship
Line		Connection
Double Ellipse		Multivalued Attribute
Double Rectangle		Weak Entity

2.1.2 Components of ER Diagram

Components

ER diagram has three core components: **Entities**, **Attributes**, and **Relationships**. Entities represent real-world objects (like Student or Course). Attributes describe properties of entities (like RollNo, Name, Age). Relationships define how entities are connected to each other (like Enrolls, Works_In).



2.2 Entity and Entity Sets

Entity

Entity means a real-world object or concept that can be uniquely identified.

- It can be **physical** — such as a person, book, laptop.

- It can be **abstract** — such as a course, bank account, reservation.

Attributes

Every entity has a set of properties called **attributes**. At least one attribute (or a combination) must uniquely identify each entity.

Example: Person Entity

- Attributes: PersonID, Name, Email
- PersonID is unique → used for identification
- Even if two people have the same name, PersonID is different

Example: Course Entity

- Attributes: CourseID, Title, Credits
- CourseID uniquely identifies each course

Entity Set

An entity set is a **collection of similar entities** that share the same attributes.

- All students in a university → entity set **Student**
- All instructors → entity set **Instructor**
- All courses → entity set **Course**

Each entity set defines:

- What type of objects are stored
- What attributes describe them

Explanation with Example

- **Entity Set (definition)** → structure description Example: Student(ID, Name, TotalCredits)
- **Entity Set Data (instances)** → actual stored rows Example: (101, Ravi, 84), (102, Meena, 72)

Entity Sets Need Not Be Disjoint

One entity can belong to multiple entity sets.

- A person may be both a **Student** and an **Instructor**
- So entity sets can overlap

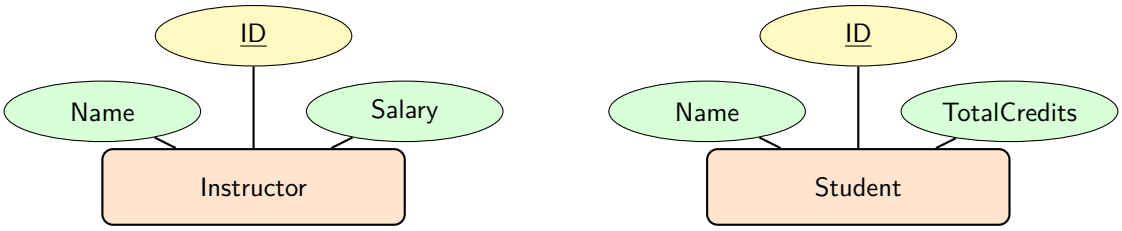
Attributes

Attributes are descriptive properties of entities.

- Each entity has one value for each attribute
- Different entities → different values

Example: Instructor Attributes

- ID = 12121
- Name = Dr. Virat
- Department = Finance
- Salary = 90000



Entity Sets with Attributes (Primary keys underlined)

2.2.1 Weak Entity Set

Weak Entity Set

A **weak entity set** is an entity set that does not have a complete primary key of its own. It cannot be uniquely identified using only its attributes.
To uniquely identify a weak entity, we must use:

- its own partial key, and
- the primary key of a related strong (owner) entity.

Because of this dependency, a weak entity:

- cannot exist independently
- must be associated with a strong entity
- is connected using an **identifying relationship**

Key Terms

- Strong Entity → has full primary key
- Weak Entity → has only partial key
- Partial Key → uniquely identifies weak entity within owner
- Identifying Relationship → links weak entity to strong entity

Example 2: Library and Book Copies

Consider a library system.
LibraryBook is a strong entity:

- BookID (primary key)
- Title
- Author

Each physical copy of a book is stored as **BookCopy**.
But CopyNumber alone is not unique globally:

- CopyNumber = 1 exists for many books

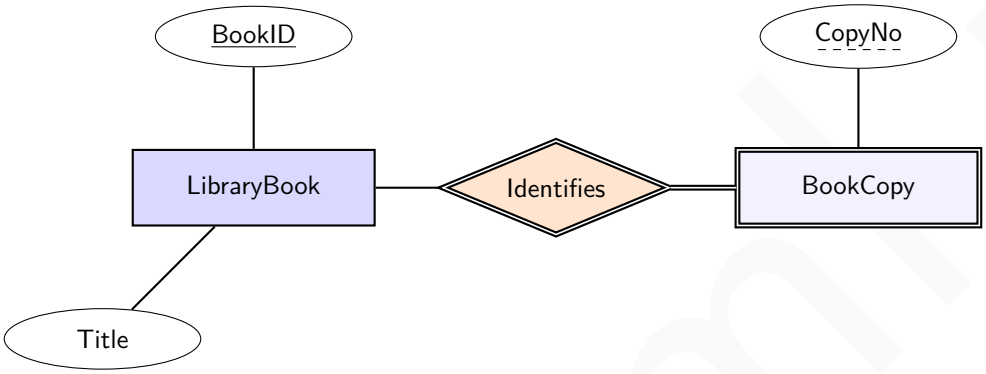
So BookCopy is a weak entity.
It is uniquely identified using:

$$(BookID + CopyNumber)$$

ER Diagram Notation Rules for Weak Entity

- Weak entity → double rectangle
- Identifying relationship → double diamond
- Partial key → dashed underline
- Total participation → double line to relationship

ER Diagram — Weak Entity Example



Weak Entity BookCopy identified by LibraryBook

2.2.2 Identifying the Entity

Concept

The process of identifying entities involves analyzing the requirements of the database system and identifying the **nouns** present in the problem statement or domain description. These nouns represent the entities that need to be represented and managed within the database. By extracting and listing these nouns, designers can establish a foundational understanding of the entities that will form the basis of the database schema.

Example 3:

Consider the goal statement for a hypothetical student enrollment database:
Goal: To assist **universities** in tracking enrolled **students**, their chosen **subjects**, and the **instructors** of those subjects.
From this statement, the following nouns can be identified as entities:

- University
- Student
- Subject
- Teacher

These entities represent the primary objects or concepts that the database will manage and track information about. It's important to note that entities are typically named using singular nouns for consistency and clarity. Additionally, the list of entities may evolve as the database design process progresses, allowing for the addition of new entities as needed to meet the requirements of the system.

2.3 Relationship Sets

Relationship Sets

A **relationship** describes how two or more entities are connected in the real world. While entities represent *objects*, relationships represent *associations between those objects*.
Think of entities as nouns and relationships as verbs.

- Entity → Customer, Product, Employee
- Relationship → buys, manages, assigned_to

A single connection between specific entities is called a **relationship instance**.

Example 4: Online Store

Assume we have two entity sets:

- Customer
- Product

If customer **Anita** buys product **Laptop**, that is one relationship instance of the relationship **purchases**.
If many customers buy many products, all such connections together form the **relationship set purchases**.

Key Idea:

Entity Set = collection of objects
Relationship Set = collection of connections between objects

Relationship Set — Formal Definition

A relationship set can be defined formally as a mathematical relation involving $n \geq 2$ entity sets (the sets need not be distinct). If the entity sets are $E_1, E_2, \ldots, E_n$, then a relationship set R consists of tuples formed by selecting one entity from each set.
In other words, R is a subset of all possible combinations:

$$(e_1, e_2, \ldots, e_n) \text{ where } e_1 \in E_1,\ e_2 \in E_2,\ \ldots,\ e_n \in E_n$$

Each such tuple is called a **relationship instance**. When entities from these sets appear in the relationship set, we say the entity sets **participate** in that relationship.

2.3.1 Degree (or arity) of a Relationship Set

Degree/Arity of a Relationship Set

The number of different entity sets participating in a relationship set is called the degree of a relationship set.

Unary OR Recursive Relationship (Degree 1)

A **unary relationship** (also called a recursive relationship) occurs when an entity set is related to itself. Here, different instances of the same entity type participate in a relationship with different roles.

Example 5: Company Employees

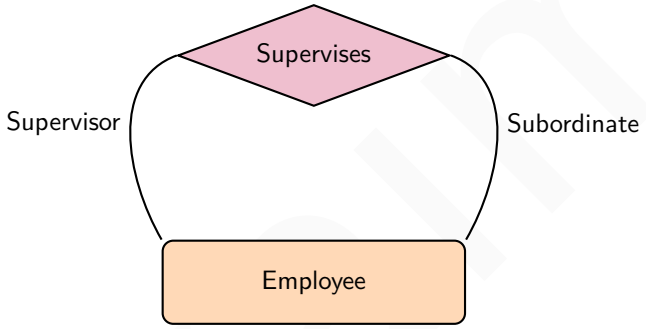
Employees can supervise other employees.

- Entity set: Employee
- Relationship: supervises
- Roles: Supervisor and Subordinate

Without role labels, the meaning becomes ambiguous.

E–R Diagram — Recursive Relationship

Recursive Relationship — Role-Based Representation



Recursive Relationship — Single Entity Representation

Binary Relationship (Degree 2)

Binary Relationship (Two Entity Sets) When exactly two entity sets are involved, it is called a **binary relationship**. This is the most common case in database design.

Example 6: Hospital System

- Entity sets: Doctor, Patient
- Relationship: treats

Meaning: A doctor treats a patient.
Each doctor–patient pair is one relationship instance.
Participation Meaning
If an entity set is connected to a relationship, we say it **participates** in that relationship.
In the example:

- Doctor participates in Treats
- Patient participates in Treats

Participation simply means "takes part in the association."

E–R Diagram — Binary Relationship

Binary Relationship: Doctor treats Patient

Ternary Relationship (Degree 3)

Relationship Involving Three Entity Sets
Some real-world facts depend on three entities together. These are modeled using a **ternary relationship**.

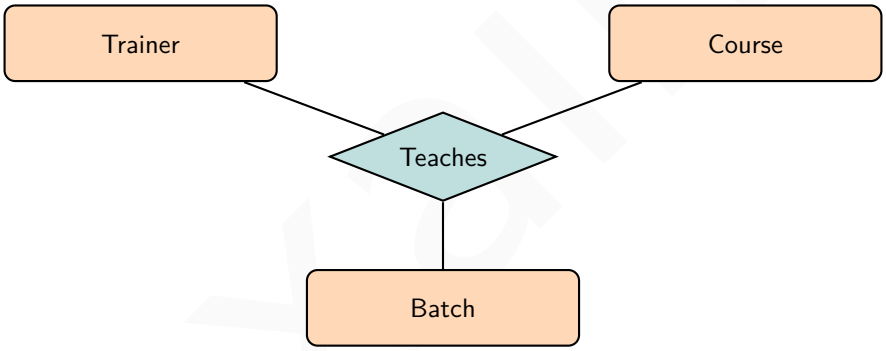
Example 7: Training System

We want to store:

- which Trainer
- teaches which Course
- to which Batch

This cannot be correctly represented using only pairs — all three must appear together.

E–R Diagram — Ternary Relationship



Ternary Relationship Example

N-ary Relationship (Degree N)

Relationship Involving N Entity Sets
When a relationship depends on **more than two entities together**, it is called an **N-ary relationship**. The relationship instance is defined only when all participating entities are considered simultaneously. Binary decomposition may lose semantic constraints in such cases.

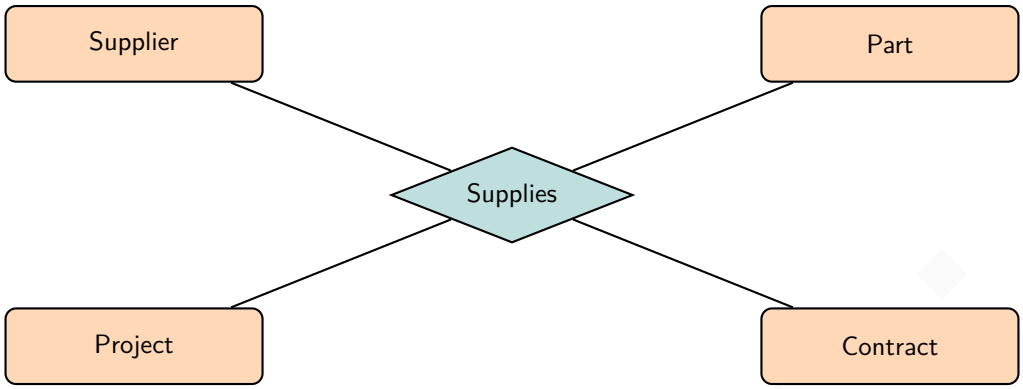
Example 8: Supply System

We want to store:

- which Supplier
- supplies which Part
- to which Project
- using which Contract

This fact depends on all entities together — so it must be modeled as an N-ary relationship.

E–R Diagram — N-ary Relationship



N-ary Relationship Example

Summary

- Relationship = association between entities
- Relationship instance = one specific connection
- Relationship set = all similar connections
- Binary = two entity sets
- Recursive = same entity set appears twice (use roles)
- Ternary = three entity sets participate together
- Diagram symbol for relationship = diamond

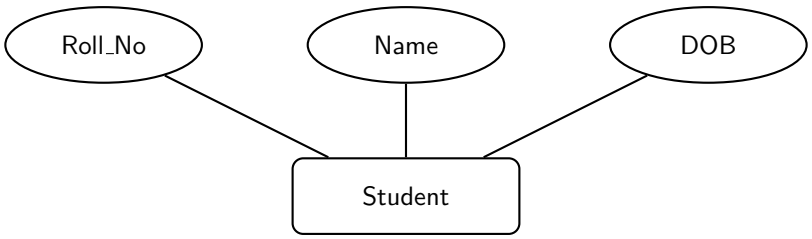
2.4 Attributes

Attributes in ER Diagram

Attributes are the descriptive properties that tell us more about an entity type. They specify what information we store about each entity instance. For example, for the entity *Student*, attributes can include Roll_No, Name, DOB, Age, Address, and Mobile_No. In an ER diagram, attributes are drawn using an **oval** connected to the entity rectangle.
Attributes are classified based on structure and behavior into key, composite, multivalued, and derived attributes.

Example 9: Basic Attribute — Student

Entity: Student with simple attributes.

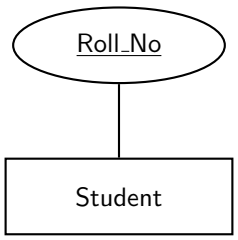


Key Attribute

A key attribute is an attribute whose value is **unique for every entity instance**. It is used to distinguish one entity from all others in the same entity set. No two entities can share the same key value. In ER diagrams, a key

attribute is shown by **underlining** the attribute name.

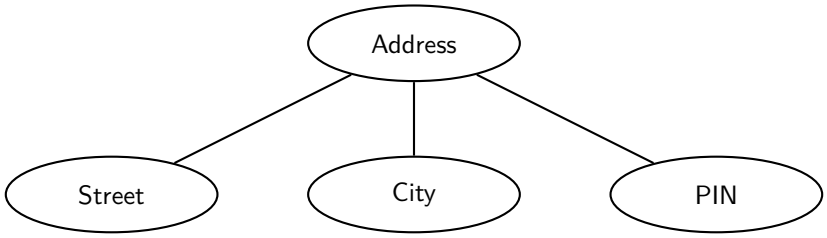
Example 10: Key Attribute — Roll Number



Composite Attribute

A composite attribute is an attribute that can be broken into smaller meaningful sub-attributes. Instead of storing it as one single value, it is modeled as a group of related parts. This helps in better structuring and querying of data. In ER diagrams, it is drawn as an oval connected to multiple sub-ovals.

Example 11: Composite Attribute — Address



Multivalued Attribute

A multivalued attribute can hold **more than one value** for a single entity instance. For example, a student may have multiple phone numbers. In ER diagrams, it is represented using a **double oval**.

Example 12: Multivalued Attribute — Phone Numbers

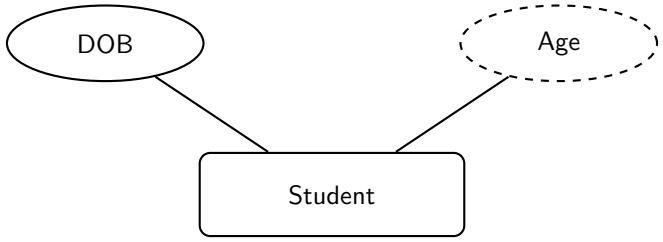
Derived Attribute

A **derived attribute** is an attribute whose value is calculated from other stored data instead of being stored directly in the database. In an ER diagram, it is always shown using a **dashed oval**. A derived attribute can be attached to either an **entity set** or a **relationship set**, depending on what it is computed from.
Rule:

- If the value is computed from attributes of one entity → attach to the **entity**.
- If the value is computed from relationship data (or multiple entities) → attach to the **relationship**.

Example 13: Derived Attribute on Entity — Age

Age is computed from DOB, so Age is derived and connected to the entity.



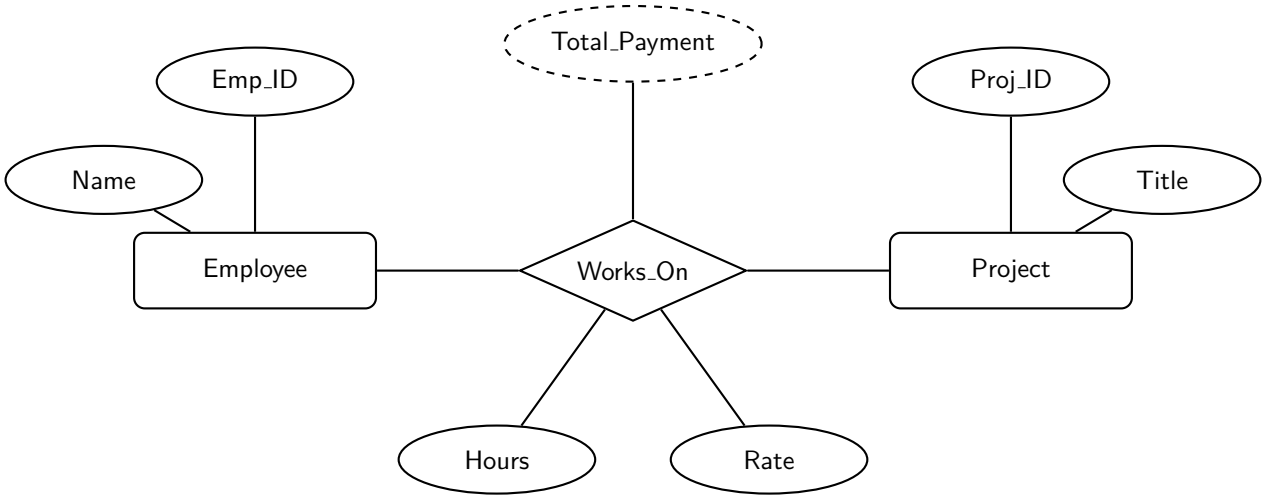
Here the dashed oval shows Age is derived from DOB.

Example 14: Derived Attribute on Relationship — Total_Payment

Total_Payment is a **derived attribute** computed as:

$$\textbf{Total\_Payment} = \textbf{Hours} \times \textbf{Rate}$$

Since it depends on relationship attributes, it is attached to the relationship diamond.



Explanation:

- **Hours** and **Rate** are attributes of the **Works_On** relationship.
- **Total_Payment** is derived as **Hours** × **Rate**.
- Therefore, the dashed oval is connected to the **relationship**, not to an entity.

Understanding Null Values in ER Modeling

A **Null Value** is a special marker used in a database to indicate that a data value does not exist in the database. It is important to distinguish between the different semantic meanings of a null value:

- **Not Applicable:** The attribute does not apply to this specific entity. *Example:* A `middle_initial` attribute for a person who does not have a middle name, or an `apt_number` (Apartment Number) for a single-family house.
- **Unknown (Missing):** The value exists in the real world, but it has not been recorded in the system. *Example:* Every `instructor` must have a `name`; if the field is null, the data is simply missing.
- **Unknown (Not Known):** It is currently unknown whether a value even exists for the entity. *Example:* If an instructor's `apt_number` is null, it could mean they don't live in an apartment (*not applicable*) OR they do live in one but we don't have the number (*missing*).

In ER diagrams, null values are not explicitly drawn as nodes; rather, they are handled by defining the **Participation Constraints** and **Nullability** of the attributes during the schema mapping phase.

2.5 Cardinality

Cardinality in ER Diagram

Cardinality specifies how many instances of one entity can be associated with instances of another entity through a relationship. It captures the numeric participation constraint between entity sets in a relationship. In ER diagrams, cardinality is shown using labels such as 1, N, or M on the connecting lines.
The four standard cardinality types are: One–to–One, One–to–Many, Many–to–One, and Many–to–Many.

Cardinality Notations Used in ER Diagrams

1. **Numeric (Ratio) Notation — Chen Style:**
   - One-to-One: 1 : 1
   - One-to-Many: 1 : N
   - Many-to-One: N : 1
   - Many-to-Many: M : N
2. **Min–Max (Participation / Multiplicity) Notation:**
   - **Exactly one:** $(1, 1)$ or $1..1$ (often simplified to just 1)
   - **Zero or one:** $(0, 1)$ or $0..1$
   - **One or many:** $(1, N)$ or $1..*$ (or $1..N$)
   - **Zero or many:** $(0, N)$ or $0..*$ (or $0..N$)
3. **Crow's Foot Notation:**
   - $|$ = one (mandatory)
   - $||$ = exactly one
   - $\circ$ = zero (optional)
   - crow foot = many

Crow's Foot Notation

Zero: The symbol above denotes zero in , indicated by the circle at the right side of the line.

One: The symbol above denotes one, shown by a short vertical line crossing the horizontal line.

Many: The symbol above denotes many; it looks like a crow's foot.

Zero or Many: This symbol combines zero and many indicators.
One or Many: This symbol combines one and many indicators.

One and Only One: This symbol indicates exactly one.

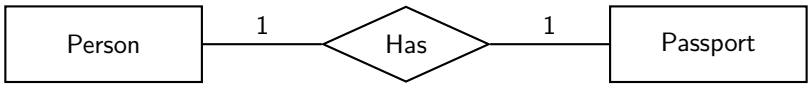
One-to-One (1:1) Cardinality

In a one-to-one relationship, each entity in set A is related to at most one entity in set B, and vice versa. This model is used when the pairing is exclusive on both sides.
Mapping note: Usually implemented using two tables with a foreign key (or sometimes merged into one table).

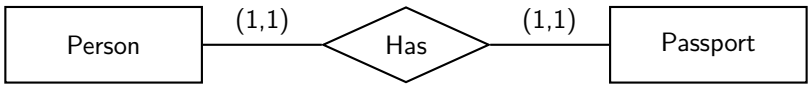
Example 15: Person — Passport : Alternate Cardinality Notations

Each Person has exactly one Passport and each Passport belongs to exactly one Person. Below are equivalent ER representations using different cardinality notations.

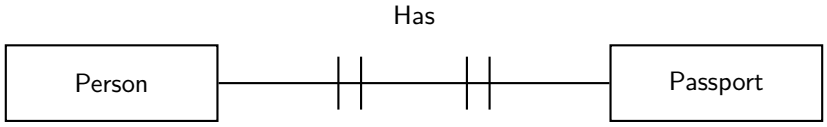
(1) Numeric Ratio Notation — 1 : 1



(2) Min–Max Notation — (1,1) : (1,1)



(3) Crow's Foot Style — exactly one on both sides



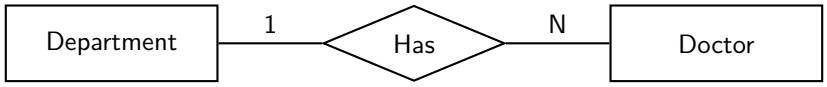
One-to-Many (1:N) Cardinality

In a one-to-many relationship, one entity from set A can be related to many entities in set B, but each entity in set B is related to only one entity in set A.
Mapping note: Implemented using two tables — foreign key placed on the "many" side.

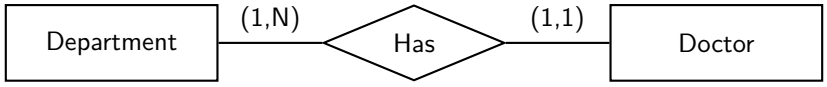
Example 16: Department — Doctor : Alternate Cardinality Notations

One Department has many Doctors, but each Doctor belongs to only one Department. Below are equivalent ER representations using different cardinality notations.

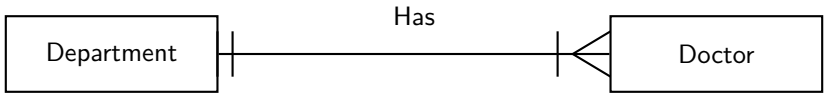
(1) Numeric Ratio Notation — 1 : N



(2) Min–Max Notation — (1,N) : (1,1)



(3) Crow's Foot Style — One-to-Many



Many-to-One (N:1) Cardinality

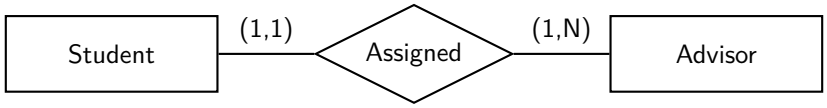
Many-to-one is the reverse view of one-to-many. Many entities in set A are related to one entity in set B. Each A chooses only one B, but B can be linked with many A's.
Mapping note: Also implemented using two tables with foreign key on the many side.

Example 17: Student — Advisor : Alternate Cardinality Notations

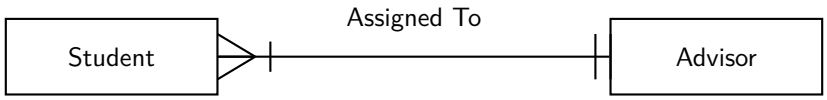
Many Students are assigned to one Advisor, but each Student has only one Advisor. Below are equivalent ER representations using different cardinality notations.

(1) Numeric Ratio Notation — N : 1

(2) Min–Max Notation — (1,1) : (1,N)



(3) Crow's Foot Style — Many-to-One



Many-to-Many (M:N) Cardinality

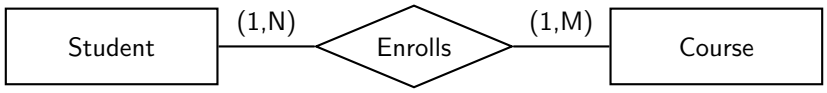
In a many-to-many relationship, entities on both sides can participate multiple times. Each entity in A can relate to many in B, and each in B can relate to many in A.
Mapping note: Requires a separate relationship table (junction table), so total tables become three.

Example 18: Student — Course : Alternate Cardinality Notations

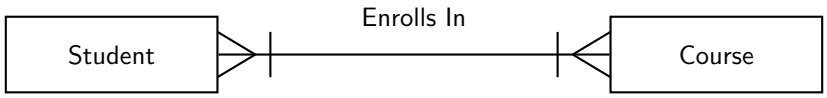
A Student can enroll in many Courses, and each Course can have many Students. Below are equivalent ER representations using different cardinality notations.

(1) Numeric Ratio Notation — M : N (or * : *)

(2) Min–Max Notation — (1,N) : (1,M)



(3) Crow's Foot Style — Many-to-Many (Mandatory Many)



2.6 Participation Constraints

Participation

In the Entity-Relationship (E-R) model, the **participation** of an entity set in a relationship set specifies whether all or only some entities in that entity set are involved in the relationship.

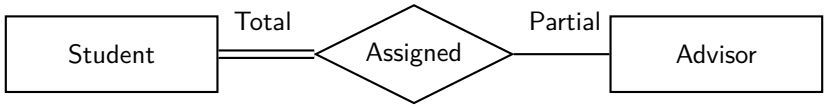
- If every entity in an entity set E must participate in at least one relationship in R, the participation is called **total**.
- If it is possible that some entities in E do not participate in any relationship in R, the participation is called **partial**.

Total Participation

Total participation occurs when **every entity** in an entity set must participate in at least one relationship in the relationship set. Total participation is represented by a double line in ER diagrams.

Example 19: Total Participation — Student and Advisor

In a university, every **Student** must have at least one **Advisor**. Therefore, the participation of the *Student* entity set in the *Assigned* relationship is total.

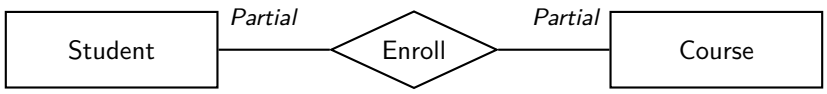


Partial Participation

Entities may or may not participate in the relationship. Partial participation is represented as single line.

Example 20: Partial Participation

In this scenario, enrollment is optional for both parties: a **Student** may exist without being enrolled in any **Course**, and a **Course** may exist without any **Students**.



2.7 Solved Examples

Example 21:

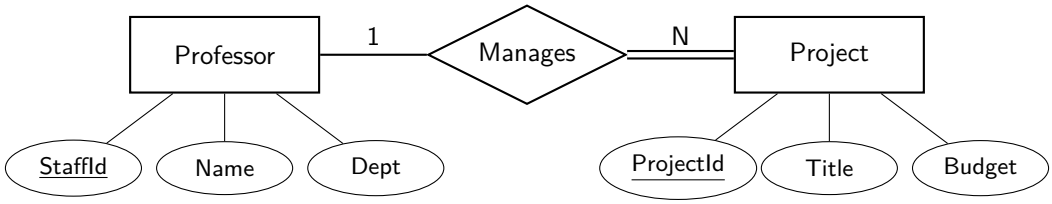
Problem Statement: A University wants to manage its research projects. Design an ER diagram based on the following business rules:

1. **Professors** are identified by a *StaffId*, *Name*, and *Dept*.
2. **Research Projects** are identified by a *ProjectId*, *Title*, and *Budget*.
3. Each Project must be managed by exactly one Professor. A Professor may manage multiple projects, but some professors may not manage any projects at all.

Solution Analysis:

- **Cardinality:** $1 : N$ (One Professor manages many Projects).
- **Participation:** **Total** on the Project side (Every project needs a manager), **Partial** on the Professor side.

ER Diagram:



Explanation: The 1 is placed near Professor and N near Project to show the 1-to-many relationship. The double line is on the Project side because a Project cannot exist without a Professor managing it (Total Participation).

Example 22:

Problem Statement: A hospital requires a database to track its operations. Based on the following requirements, identify the entities, relationships, cardinalities, and participation constraints, and provide an ER diagram:

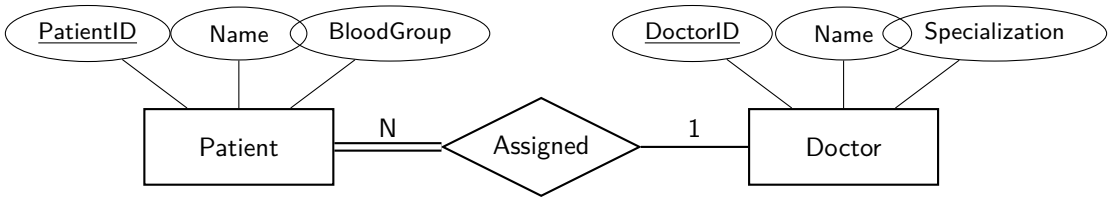
1. **Patients** are identified by a *PatientID*, *Name*, and *BloodGroup*.
2. **Doctors** are identified by a *DoctorID*, *Name*, and *Specialization*.
3. Each patient is assigned to exactly one primary doctor for their stay. A doctor can be the primary doctor for many patients, but some doctors might not have any primary patients assigned to them.
4. A patient must have at least one doctor assigned to them to be admitted.

Solution:
1. Constraint Analysis:

- **Cardinality:** $N : 1$ (Many Patients to one Doctor).

- **Participation: Total** for Patient (Double line), **Partial** for Doctor (Single line).

2. ER Diagram:



Explanation: The double line connects *Patient* to the relationship because the problem states every patient **must** have a doctor. The single line for *Doctor* reflects that some doctors do not have assigned patients.

Example 23:

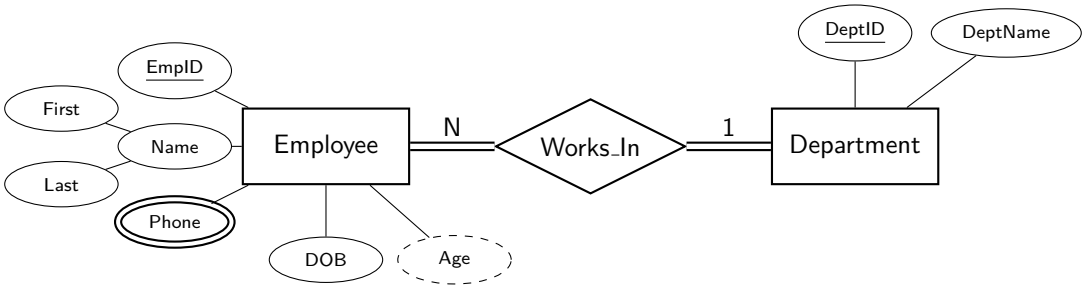
Problem Statement: A Software Company needs a database to manage its employees. Design an ER diagram based on the following requirements:

1. **Employees** have an *EmployeeID* (unique), a *Salary*, and a *Name*. The Name consists of a *First Name* and a *Last Name*.
2. We track the *Phone Numbers* of employees; an employee can have more than one phone number.
3. We store the *Date of Birth* of each employee and need to display their *Age* in the system.
4. Employees work in **Departments**. A department has a *DeptID* and a *DeptName*.
5. Every employee must belong to exactly one department. A department must have at least one employee to exist in the system.

Solution Analysis:

- **Composite Attribute:** *Name* (split into First and Last Name).
- **Multi-valued Attribute:** *Phone Number* (represented by a double ellipse).
- **Derived Attribute:** *Age* (calculated from Date of Birth, represented by a dashed ellipse).
- **Cardinality:** $1 : N$ (One Department has many Employees).
- **Participation: Total** on both sides (Every employee needs a department, and every department needs at least one employee).

ER Diagram:



Explanation:

- **Double Ellipse (Phone):** Indicates a multi-valued attribute.

- **Dashed Ellipse (Age):** Indicates a derived attribute because age changes over time and is calculated from the Date of Birth.
- **Hierarchical Attributes (Name):** Represents a composite attribute.
- **Double Lines:** Indicates that the relationship is mandatory for both entities (Total Participation).

Example 24:

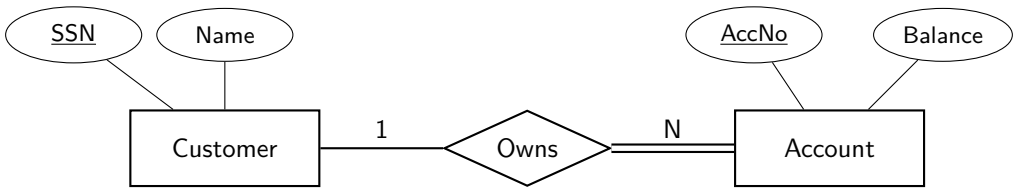
Problem Statement: A Bank manages customer accounts. Design an ER diagram based on the following rules:

1. **Customers** have a *SSN* (unique), a *Name*.
2. **Accounts** have an *AccountNumber* and a *Balance*.
3. Every **Account** must be owned by exactly one **Customer**.
4. A **Customer** can own multiple accounts, but a person can be a customer of the bank without having an account yet (e.g., they just have a loan or a profile).

Solution Analysis:

- **Cardinality:** $1 : N$ (One Customer can have many Accounts).
- **Customer Participation (Partial):** A Customer can exist with zero accounts. We use a **Single Line**.
- **Account Participation (Total):** An Account cannot exist without an owner. We use a **Double Line**.

ER Diagram:



Why the lines are drawn this way:

- **The '1' and 'N':** These tell us the **Maximums**. One customer → Many accounts (N). One account → One customer (1).
- **The Double Line (on Account side):** This tells us the **Minimum**. Every account *must* have at least one owner. It is a "Mandatory" relationship for the Account.
- **The Single Line (on Customer side):** Participation is **Optional**. A Customer *can* exist in the database without being linked to an account.

Example 25:

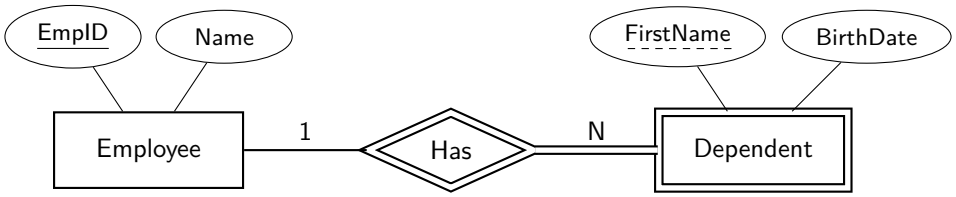
Problem Statement: An Insurance Company tracks employees and their dependents.

1. **Employees** have an *EmpID* (unique) and a *Name*.
2. **Dependents** (family members) have a *FirstName* and *BirthDate*.
3. Dependents do not have a unique ID of their own; they are identified by their *FirstName* combined with their specific Employee's ID.
4. Each Dependent belongs to exactly one Employee. An Employee can have zero or more dependents.

Solution Analysis:

- **Weak Entity:** *Dependent* is a weak entity because it lacks a primary key.
- **Identifying Relationship:** The link between Employee and Dependent is an "Identifying Relationship" (drawn with a double diamond).
- **Partial Key:** *FirstName* is a partial key (discriminator), drawn with a dashed underline.
- **Participation:** Total on the Dependent side (A dependent must have an employee to exist).

ER Diagram:



2.8 Problems

Problem 34 *Draw an ER diagram to represent: "a customer can place many orders but a order cannot be placed by many customers"*

Problem 35 *Draw an ER diagram to represent: "many students can study in a single college but a student cannot study in many colleges at the same time"*

Problem 36 *An Online Learning Platform needs a database to manage its academic activities. Design an ER diagram based on the following requirements:*

1. ***Courses** are identified by a CourseID and have Title, Duration, and Fee.*
2. *Each Course is divided into multiple **Modules**. A Module has ModuleNo, Name, and Hours. Module numbers are unique only within a course.*
3. *A Module cannot exist without its Course.*
4. ***Instructors** are identified by InstructorID and have Name and Expertise.*
5. *An Instructor can teach many Courses and a Course can be taught by multiple Instructors.*
6. *For each teaching assignment, store the AssignedDate.*
7. *Each Instructor may have multiple **Email** addresses stored in the system.*
8. *The ExperienceYears of an Instructor is derived from their JoiningDate.*
9. *Some Courses may not yet be assigned to any Instructor.*

Problem 37 *A Logistics Company wants to design a database for shipment tracking. Construct an ER diagram from the following rules:*

1. ***Warehouses** are identified by WarehouseID and have Location and Capacity.*
2. *Each Warehouse contains many **StorageUnits**. A StorageUnit has UnitNo and Type. Unit numbers are unique only within a warehouse.*
3. *A StorageUnit cannot exist unless it belongs to a Warehouse.*
4. ***Shipments** are identified by ShipmentID and have Weight and DispatchDate.*
5. *Each Shipment is stored in exactly one StorageUnit, but a StorageUnit may store many Shipments.*
6. *Every Shipment must be stored in some StorageUnit.*
7. ***Vehicles** are identified by VehicleNo and have Model.*
8. *A Vehicle transports many Shipments, but each Shipment is transported by exactly one Vehicle.*

Problem 38 *A Conference Management System is to be modeled using an ER diagram. Use the following constraints:*

1. ***Conferences** are identified by ConfID and have Name, City, and StartDate.*
2. *Each Conference has multiple **Sessions**. A Session has SessionNo, Topic, and Hall. Session numbers are unique within a conference.*
3. *A Session cannot exist without its Conference.*
4. ***Speakers** are identified by SpeakerID and have a composite Name (First, Last) and Affiliation.*
5. *Speakers can deliver many Sessions and a Session can have multiple Speakers.*
6. *For each speaker–session association, record the Duration.*
7. *Speakers may have multiple **ContactNumbers**.*

8. *The Age of a Speaker is derived from DateOfBirth.*

Problem 39 *An ER diagram contains an entity "Book" with attributes "ISBN", "Title", and "Authors". Since a book can have multiple authors, "Authors" is a multi-valued attribute. If we want to store the "Rank" of each author for a specific book, what is the best design choice?*
A. Add "Rank" as an attribute to the "Book" entity.
B. Add "Rank" as an attribute to the "Author" entity.
C. Create a relationship "WrittenBy" and add "Rank" as a descriptive attribute to the relationship.
D. Make "Rank" a derived attribute of "Book".

Problem 40 *A weak entity set can always be made into a strong entity set by adding to its attributes the primary key. In this context, choose the statement(s) that are correct.*
A. This may lead to redundancy.
B. If we add primary-key attributes to the weak entity set, they will be present in both the entity set and the relationship set, and they need not be the same.
C. The primary key of a weak entity set can be inferred from its relationship with the strong entity set.
D. If we add primary-key attributes to the weak entity set, they will be present in both the entity set and the relationship set, and they have to be the same.

Problem 41 *In an ER diagram, if entity set X has **Total Participation** in a relationship R with entity set Y, which of the following must be true for every instance $x \in X$?*
A. There exists at least one instance $y \in Y$ such that $(x, y) \in R$.
B. There exists at most one instance $y \in Y$ such that $(x, y) \in R$.
C. There exists exactly one instance $y \in Y$ such that $(x, y) \in R$.
D. X must be a weak entity set.

Problem 42 *A "Customer" entity has an attribute "EmailAddresses". A customer may have no email, one email, or multiple emails. This attribute is best classified as:*
A. Simple and Single-valued
B. Composite and Multi-valued
C. Simple and Multi-valued
D. Derived and Single-valued

Problem 43 *Consider the rule: "An employee can manage at most one department, and each department is managed by exactly one employee." What are the cardinalities for Employee and Department in the "Manages" relationship?*
A. $1 : 1$
B. $1 : N$
C. $N : 1$
D. $M : N$

Problem 44 *Which of the following describes a **Derived Attribute**?*
A. An attribute that can be divided into smaller sub-parts.
B. An attribute whose value is calculated from other attributes or entities.
C. An attribute that uniquely identifies an entity.
D. An attribute that can have more than one value for a single entity.

Problem 45 *Study the following diagram:*

What does the double line on the "Book" side imply?
A. Every Author must write at least one Book.
B. Every Book must have at least one Author.
C. Authors can exist without having written any Books.
D. Both B and C are correct.

Problem 46 *A Weak Entity set is distinguished by the fact that:*
A. It has no primary key of its own and depends on an identifying entity.
B. It is always on the '1' side of a $1:N$ relationship.
C. It has partial participation in its identifying relationship.
D. It can exist independently of its identifying entity set.

Problem 47 *A "Name" attribute consisting of "First_Name", "Middle_Initial", and "Last_Name" is a:*
A. Simple Attribute
B. Composite Attribute
C. Multi-valued Attribute
D. Derived Attribute

Problem 48 *In an ER model, a "Null" value for the attribute "ApartmentNumber" in an address entity most likely represents:*
A. A missing value that exists but is not recorded.
B. A value that is "not applicable" for that entity.
C. An unknown value.
D. Any of the above, depending on the specific entity instance.

Problem 49 *An attribute like "Hours_Worked" in a relationship between "Employee" and "Project" is called a:*
A. Key Attribute
B. Descriptive Attribute
C. Derived Attribute
D. Partial Key

Problem 50 *An object should be modeled as an **Entity** instead of an **Attribute** if:*
A. It has its own descriptive attributes and needs to be related to other entities.
B. It represents a numeric value like Salary.
C. It is only used once in the entire system.
D. It is a simple string value that does not require further description.

Problem 51 *If every Instructor must belong to exactly one Department, but a Department can have zero or more Instructors, the relationship from Instructor to Department is:*
A. $1:N$ with partial participation of Instructor.
B. $N:1$ with total participation of Instructor.
C. $1:1$ with total participation of Department.
D. $M:N$ with partial participation of both.

Problem 52 *Total participation of entity set E in relationship R implies that the minimum cardinality of E in R is:*
A. 0
B. 1
C. N
D. Undefined

Problem 53 *In a $1 : N$ relationship R between entity A and B (where A is the 1-side), a single entity in B can be associated with:*
A. At most one entity in A.
B. At least one entity in A.
C. Multiple entities in A.
D. Exactly N entities in A.

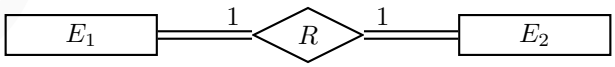
Problem 54 *In a Library system, "Total_Books_Borrowable" for a student is determined by checking their membership level and current loans. This attribute is:*
A. Multi-valued
B. Composite
C. Derived
D. Simple

Problem 55 *In a ternary relationship R between entity sets A, B, and C, a cardinality constraint of $(1, 1)$ is placed on the edge connecting A to R. This specific constraint implies which of the following regarding the relationship?*
A. For any pair of entities (b, c) from $B \times C$, there is exactly one entity $a \in A$ associated with them in R.
B. Every entity $a \in A$ must be associated with exactly one pair (b, c) in R.
C. The relationship R can be decomposed into two binary relationships without loss of information.
D. Entity A is a weak entity set identifying with B and C.

Problem 56 *Consider an entity set **Account** and a relationship **Transfers** that connects **Account** to itself (recursive). The relationship represents a transfer from a "Source" account to a "Destination" account. If an account can be a source for many transfers but a destination for at most one, what is the cardinality of the relationship?*
A. $1 : 1$
B. $1 : N$
C. $M : N$
D. $N : 1$

Problem 57 *In an ER diagram, an attribute A of entity set E is changed from being **Single-valued** to **Multi-valued**. Which of the following structural changes occurs in the conceptual model?*
A. The entity set E must now be modeled as a weak entity set.
B. The degree of the relationship involving E increases by one.
C. The primary key of E may no longer be sufficient to identify the values of A.
D. The participation of E in any relationship R must become total.

Problem 58 *Consider the following ER diagram:*



Which statement regarding the existence dependency is correct based on this diagram?
A. E_1 can exist without E_2, but E_2 cannot exist without E_1.
B. E_1 and E_2 must have a one-to-one correspondence, and neither can exist in the database without the other via relationship R.
C. This represents a partial key relationship where E_2 is a weak entity.
D. The relationship R allows an instance of E_1 to be related to multiple instances of E_2.

Problem 59 *An attribute A is **Composite** and one of its components A_1 is **Multi-valued**. In a university database, this could represent which of the following?*
A. A student's GPA which is calculated every semester.
B. A student's Address where Phone_Numbers is one of the fields.
C. A student's Name which consists of First and Last.

D. A student's ID which is a unique number.

Problem 60 *If an entity set E participates in a relationship R with a **Max Cardinality** of 1, and the participation is **Partial**, what is the range of the number of relationship instances an entity $e \in E$ can participate in?*
A. Exactly 1
B. 0 or 1
C. 1 or more
D. 0 to N

Problem 61 *Which of the following scenarios describes an **Identifying Relationship**?*
A. A relationship between two strong entities with $M : N$ cardinality.
B. A relationship that links a weak entity set to the entity set on which it depends for existence and identification.
C. A relationship where the primary keys of both entities are merged.
D. A recursive relationship where an employee manages themselves.

Problem 62 *In a Ternary relationship among Supplier, Part, and Project, the constraint is: "A supplier supplies a particular part to at most one project." Where should the cardinality '1' be placed?*
A. On the Supplier side.
B. On the Part side.
C. On the Project side.
D. In the center of the diamond.

Problem 63 *Which of the following is TRUE regarding a **Total Participation** constraint?*
A. It is a Max-cardinality constraint.
B. It is a Min-cardinality constraint where $Min = 1$.
C. It can only be applied to $M : N$ relationships.
D. It requires the entity to have at least two attributes.

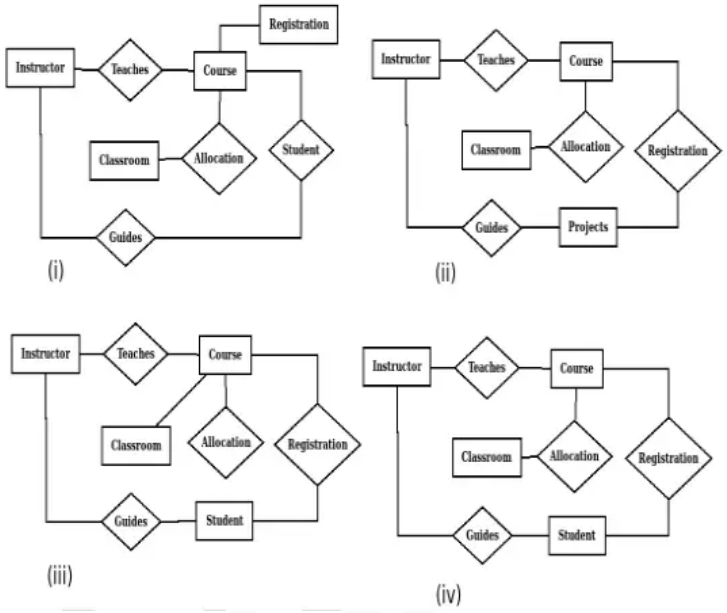
Problem 64 *If a relationship R is $1 : 1$ and participation is partial on both sides, an entity can participate in R:*
A. Exactly once.
B. At most once.
C. At least once.
D. Zero times only.

2.9 GATE PYQs

GATEPYQ 1 *In the context of owner and weak entity sets in the ER (Entity–Relationship) data model, which one of the following statements is TRUE?* ***GATE CSE 2024***

A. The weak entity set MUST have total participation in the identifying relationship
B. The owner entity set MUST have total participation in the identifying relationship
C. Both weak and owner entity sets MUST have total participation in the identifying relationship
D. Neither weak entity set nor owner entity set MUST have total participation in the identifying relationship

GATEPYQ 2 *Let S be the specification: "Instructors teach courses. Students register for courses. Courses are allocated classrooms. Instructors guide students."* ***GATE CSE 2024***



A. (i)
B. (ii)
C. (iii)
D. (iv)

GATEPYQ 3 *Which one of the following is used to represent the supporting many-one relationships of a weak entity set in an entity-relationship diagram?* ***Gate 2020***

A. Diamonds with double/bold border
B. Rectangles with double/bold border
C. Ovals with double/bold border
D. Ovals that contain underlined identifiers

GATEPYQ 4 *In an Entity–Relationship (ER) model, suppose R is a many-to-one relationship from entity set E1 to entity set E2. Assume that E1 and E2 participate totally in R and that the cardinality of E1 is greater than the cardinality of E2. Which one of the following is true about R?* ***GATE CSE 2018***

A. Every entity in E1 is associated with exactly one entity in E2.
B. Some entity in E1 is associated with more than one entity in E2.
C. Every entity in E2 is associated with exactly one entity in E1.
D. Every entity in E2 is associated with at most one entity in E1.

GATEPYQ 5 *Which symbol denotes derived attributes in ER Model?* ***Gate 2017***

A. Double ellipse
B. Dashed ellipse
C. Squared ellipse
D. Ellipse with attribute name underlined

GATEPYQ 6 *Given the basic ER and relational models, which of the following is INCORRECT?* ***GATE CSE 2012***

A. An attribute of an entity can have more than one value
B. An attribute of an entity can be composite
C. In a row of a relational table, an attribute can have more than one value
D. In a row of a relational table, an attribute can have exactly one value or a NULL value

GATEPYQ 7 *Consider the entities 'hotel room', and 'person' with a many to many relationship 'lodging' as shown below.* ***Gate 2005***

If we wish to store information about the rent payment to be made by person (s) occupying different hotel rooms, then this information should appear as an attribute of

A. Person.
B. Hotel Room.
C. Lodging.
D. None of These.

2.10 Try it Yourself

Exercise 34 ***Scenario: University Hostel Management***
A university wants to model its hostel system.

- *Each **Hostel** has a unique HostelName and a Location.*
- ***Rooms** are located within Hostels. A room is identified by a RoomNumber, but only within a specific hostel (e.g., Room 101 exists in multiple hostels).*
- *Every Room must belong to exactly one Hostel.*

*In the resulting ER diagram, how should the **Room** be represented?*
A. As a strong entity with RoomNumber as the primary key.
B. As a weak entity with RoomNumber as the discriminator, connected via a double diamond.
*C. As a multi-valued attribute of the **Hostel** entity.*
*D. As a relationship attribute between **Hostel** and **Student**.*

Exercise 35 ***Scenario: E-Commerce Orders***
In an online store:

- ***Customers** (identified by Email) place **Orders**.*
- *An Order must be placed by exactly one customer.*
- *A customer can exist in the system without having placed any orders yet.*

*If you are drawing the connection between **Customer** and **Order** using Chen's notation, which of the following is correct?*
A. A single line on the Customer side and a double line on the Order side.
B. A double line on the Customer side and a single line on the Order side.
C. Double lines on both sides.
D. Single lines on both sides.

Exercise 36 ***Scenario: Project Assignments***
*A company tracks **Employees** and **Projects**.*

- *An employee can work on multiple projects.*
- *A project can have multiple employees.*
- *For every assignment, we must record the DateStarted and the Role the employee plays in that specific project.*

Where should the attributes DateStarted and Role be drawn?
*A. Attached to the **Employee** entity.*
*B. Attached to the **Project** entity.*
*C. Attached to the **WorksOn** diamond relationship.*
*D. As derived attributes of the **Department** entity.*

Exercise 37 ***Scenario: Banking Hierarchy***
*A bank has **Branches**. Each Branch has **Accounts**.*

- *Every branch must have at least one account to be active.*
- *Every account must belong to exactly one branch.*
- *One branch can manage thousands of accounts.*

*If the relationship is Manages, what is the correct cardinality and participation on the **Branch** side?*
A. Cardinality 1, Partial Participation.
B. Cardinality N, Total Participation.
C. Cardinality 1, Total Participation.
D. Cardinality N, Partial Participation.

Exercise 38 ***Scenario: Vehicle Registration***
*A **Person** owns a **Vehicle**.*

- *A person can own zero or more vehicles.*
- *Each vehicle is owned by exactly one person.*
- *The system must calculate the TotalTax owed by a person based on the number of vehicles they own.*

In the ER diagram, TotalTax should be represented as:
A. A simple key attribute.
B. A multi-valued attribute.
C. A dashed ellipse (derived attribute).
D. A partial key for a weak entity.

Exercise 39 ***Scenario: Airline Flight System***
*A **Flight** is scheduled between two **Airports** (a Departure airport and an Arrival airport). Both Departure and Arrival are the same type of entity (**Airport**). This is an example of:*
A. A Ternary Relationship.
B. A Recursive Relationship with two different roles.
C. A Weak Entity set.
D. A Multi-valued attribute of the Flight entity.

Exercise 40 ***Scenario: Employee Dependents***
*In an insurance database, an **Employee** has **Dependents** (Spouse, Children).*

- *If an employee leaves the company, their dependents' data is no longer required.*
- *Dependents are identified only by their Name and the EmployeeID of their parent/spouse.*

*Which TikZ structural element is used for the **Relationship** between Employee and Dependent?*
A. `\node[draw, diamond] (r) {Has};`
B. `\node[draw, diamond, double] (r) {Has};`
C. `\node[draw, rectangle, double] (r) {Has};`
D. `\node[draw, ellipse, dashed] (r) {Has};`

Exercise 41 ***Scenario: Pharmacy Database***
*A **Drug** is sold by various **Pharmacies**.*

- *Each drug has a unique TradeName.*
- *Different pharmacies sell the same drug at different Prices.*

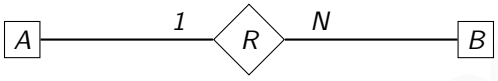
How should Price be modeled in the ER diagram?
*A. As a primary key for the **Drug** entity.*
*B. As an attribute of the **Pharmacy** entity.*
*C. As an attribute of the **Sells** relationship between Drug and Pharmacy.*
*D. As a composite attribute of the **Drug** entity.*

Exercise 42 *In an ER diagram, if the entity set E has a total participation constraint in a binary relationship R with entity set F, which of the following is always TRUE?*
A. The cardinality of R must be $1:N$.
B. Every entity in E must be associated with at least one entity in F through R.
C. The relationship R must have a descriptive attribute.
D. E must be a weak entity set.

Exercise 43 *A "Multi-valued" attribute is represented in Chen's notation by a double ellipse. If an entity "Person" has a multi-valued attribute "Hobbies", and a specific person has no hobbies, how is this represented in the conceptual model?*
A. The attribute takes a NULL value.
B. The entity instance cannot exist.
C. The attribute is omitted for that specific instance.
D. The attribute is converted into a derived attribute.

Exercise 44 *Consider a relationship R between E_1 and E_2. The Min-Max notation on the E_1 side is $(1, 1)$. This implies:*
A. Partial participation and $1:1$ cardinality.
B. Total participation and $1:1$ or $N:1$ cardinality.
C. Total participation and $1:N$ cardinality.
D. Partial participation and $M:N$ cardinality.

Exercise 45 *In the following ER diagram:*



Which of the following describes the cardinality ratio from A to B?
A. One A can be related to many B, and one B to one A.
B. One B can be related to many A, and one A to one B.
C. Many A can be related to many B.
D. Every A must be related to at least one B.

Exercise 46 *A weak entity set W is identified by strong entity S via relationship R. If S is deleted from the database, what happens to W?*
A. W becomes a strong entity.
B. W is also deleted because of existence dependency.
C. W remains in the database but its primary key becomes NULL.
D. W is moved to a temporary archive table.

Exercise 47 *Which of the following attributes is the best example of a **Composite Attribute**?*
A. Employee ID Number.
B. Date of Birth.
C. GPS Coordinates (Latitude, Longitude).
D. Salary.

Exercise 48 *In a Ternary relationship $R(A, B, C)$, if the cardinality is $1:1:1$, how many total functional dependencies are implicitly defined among the primary keys of the entities?*
A. 1
B. 3
C. 6
D. 9

Exercise 49 *If an attribute A is **Derived**, it is usually represented by a dashed ellipse. Which of the following is a reasoning for NOT storing a derived attribute in a physical table?*
A. To save disk space only.
B. To ensure data consistency when the base attributes change.
C. Because SQL does not support derived data types.
D. Because derived attributes cannot have NULL values.

Exercise 50 *Consider a self-referencing relationship "Supervises" on the "Employee" entity. If every employee has exactly one supervisor, but a supervisor can manage many employees, what is the cardinality?*
A. $1:1$
B. $1:N$
C. $M:N$
D. $N:M$

Exercise 51 *The "Partial Key" of a weak entity set is also known as the:*
A. Primary Key.
B. Foreign Key.
C. Discriminator.
D. Surrogate Key.

Exercise 52 *In a* $1:N$ *relationship* R *between* A *and* B *(where* A *is the 1-side), total participation on the* A *side means:*
A. Every B *must be related to an* A*.*
B. Every A *must be related to at least one* B*.*
C. Every A *must be related to exactly one* B*.*
D. A *is a weak entity.*

Exercise 53 *A relationship is said to be "Recursive" when:*
A. It involves three or more entity sets.
B. The same entity set participates in the relationship more than once in different roles.
C. It connects a weak entity to its owner.
D. It has a multi-valued attribute.

Exercise 54 *Identify the* ***incorrect*** *statement regarding NULL values in ER modeling:*
A. NULL can represent an unknown value.
B. NULL can represent a "not applicable" value.
C. A primary key attribute can be NULL if the participation is partial.
D. NULL values are used when a value exists but is missing from the records.

Exercise 55 *In an ER diagram, a double-lined rectangle represents:*
A. A relationship set.
B. A weak entity set.
C. A multi-valued attribute.
D. A total participation constraint.

Exercise 56 *Which of the following is true for a* ***Simple Attribute****?*
A. It cannot be divided into sub-parts.
B. It can have multiple values for a single entity.
C. It is always a primary key.
D. It is calculated from other attributes.

Exercise 57 *In a ternary relationship* $R(A, B, C)$*, the cardinality* $(0, N)$ *on all sides implies:*
A. The relationship is mandatory for all entities.
B. The relationship is optional for all entities.
C. Any entity in A *can be related to only one pair of* (B, C)*.*
D. The relationship can be decomposed into binary relationships.

Exercise 58 *An attribute that is both* ***Composite*** *and* ***Multi-valued*** *can be exemplified by:*
A. A student's birthdate.

B. A list of previous jobs, where each job has a title and a duration.
C. A student's unique roll number.
D. A student's age.

Exercise 59 *Total participation of a strong entity in a relationship is represented by:*
A. A double ellipse.
B. A double line.
C. A dashed line.
D. A bold diamond.

Exercise 60 *In an $M : N$ relationship between A and B, how many entities from A can be associated with a single entity from B?*
A. Exactly one.
B. At most one.
C. Zero or more.
D. Exactly M.

Exercise 61 *A "Descriptive Attribute" is an attribute that:*
A. Belongs to an entity set.
B. Belongs to a relationship set.
C. Is used as a primary key.
D. Is always derived.

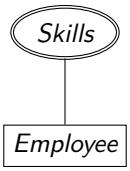
Exercise 62 *What is the degree of a relationship that connects four different entity sets?*
A. 2
B. 3
C. 4
D. N

Exercise 63 *Consider a weak entity W and its identifying relationship R with strong entity S. Which statement is true?*
A. R must be $M : N$.
B. R must be $1 : N$ or $1 : 1$ with W on the N or 1 side.
C. W can have partial participation in R.
D. S must be a weak entity too.

Exercise 64 *A "Degree" of a relationship refers to:*
A. The number of entities in the entity set.
B. The number of entity sets participating in the relationship.
C. The maximum number of relationships an entity can participate in.
D. The number of attributes in the relationship.

Exercise 65 *Which of the following cannot be a primary key?*
A. A composite attribute.
B. A derived attribute.
C. A simple attribute.
D. A single-valued attribute.

Exercise 66 *In the diagram below:*



What does "Skills" represent?
A. A derived attribute.
B. A multi-valued attribute.
C. A composite attribute.
D. A weak entity.

Exercise 67 *In an ER model, if we want to ensure that every "Loan" must be associated with at least one "Customer", we use:*
A. A $1:1$ cardinality ratio.
B. A total participation constraint on the Loan side.
C. A partial participation constraint on the Loan side.
D. A multi-valued attribute for Customer.

Exercise 68 *A "Discriminator" is used to:*
A. Identify a strong entity.
B. Identify a weak entity within the context of a strong entity.
C. Calculate derived attributes.
D. Link two $M:N$ relationships.

Exercise 69 *In an ER diagram for a hospital, "Doctor" treats "Patient". If a patient can be treated by multiple doctors and a doctor can treat multiple patients, the cardinality is:*
A. $1:1$
B. $1:N$
C. $N:1$
D. $M:N$

Exercise 70 *An attribute like "Street_Address" which can be broken down into "Zip", "City", and "Street_Name" is:*
A. Simple.
B. Composite.
C. Multi-valued.
D. Derived.

Exercise 71 *The number of entities associated with each other via a relationship is called the:*
A. Entity Count.
B. Cardinality Ratio.
C. Relationship Degree.
D. Participation Factor.

Exercise 72 *A double diamond in an ER diagram signifies:*
A. A multi-valued relationship.
B. An identifying relationship for a weak entity set.
C. A recursive relationship.
D. A ternary relationship.

Exercise 73 *In an ER diagram, the primary key of an entity set is indicated by:*
A. An asterisk.

B. A dashed underline.
C. A solid underline.
D. A double ellipse.

Exercise 74 *Which of the following is **not** a structural constraint in ER diagrams?*
A. Cardinality ratio.
B. Participation constraint.
C. Attribute data type.
D. Existence dependency.

Exercise 75 *A relationship R between A and B is $1:N$. If we move an attribute from A to R:*
A. It becomes a multi-valued attribute.
B. It remains a single-valued attribute for each relationship instance.
C. It must be converted into a primary key.
D. It is no longer allowed in ER modeling.

Exercise 76 *Participation of an entity set in a relationship is **Partial** if:*
A. Only some entities in the set participate in the relationship.
B. All entities in the set participate in the relationship.
C. The entity has no primary key.
D. The entity is a weak entity.

Exercise 77 *A "Key Attribute" must be:*
A. Unique for each entity.
B. Multi-valued.
C. Derived from other attributes.
D. Composite at all times.

Exercise 78 *In a $1:1$ relationship with total participation on both sides, the two entities:*
A. Must be merged into a weak entity set.
B. Have a mandatory existence dependency on each other.
C. Can never have descriptive attributes.
D. Must have the same primary key names.

Exercise 79 *An ER diagram is considered a:*
A. Physical Level model.
B. Conceptual Level model.
C. Internal Level model.
D. Implementation Level model.

2.11 YouTube Links and QR Codes

Lecture	Details	YouTube Link	QR Code
5	ER Model — ER Diagram — Entity & Entity Set	https://youtu.be/onO5D70qrS0	
6	Relationship Set — Degree (Unary, Binary, Ternary, N-ary) — Types of Attributes	https://youtu.be/RbspfcDqm58	
7	Cardinality Ratio & Participation Constraints	https://youtu.be/e-rNrAOF_2M	
8	Solved Examples on ER Diagrams	https://youtu.be/obbVs-euSQE	

9	Practice Problems on ER Diagrams	https://youtu.be/9PkYnKQ2-CE	
10	GATE PYQs on ER Diagrams	https://youtu.be/5ciP-BzCCuQ	

Chapter 3

Relational Model

3.1 Structure of Relational Databases

Relational Model

A **Relational Model** is fundamentally a collection of tables, where each table is assigned a unique name. In the formal mathematical world, these tables are known as **Relations**. The power of the relational model lies in its simplicity: data is represented as a set of related values organized into rows and columns.

Key Terminology

To understand the relational model, we must bridge the gap between common database language and formal mathematical terms:

- **Relation / Table:** A structure consisting of rows and columns.
- **Tuple / Row:** A single record representing a relationship among a set of values.
- **Attribute / Column:** A specific header defining a type of data within the relation.
- **Relation Instance:** A specific set of data (rows) present in a table at a given moment.

Example 26:

Consider a table used to store information about faculty. Each row represents a specific instructor identified by a unique ID.

ID	Name	Dept. Name	Salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000

Example 27:

This table defines the academic offerings. Note that the attribute `dept_name` appears here as well, creating a logical link between instructors and courses via their departments.

Course ID	Title	Dept. Name	Credits
CS-101	Intro. to Computer Science	Comp. Sci.	4
BIO-301	Genetics	Biology	4
PHY-101	Physical Principles	Physics	4

Concept

In mathematics, a **tuple** is a sequence of n values. A relationship between n values is represented as an n-**tuple**. Because a table is essentially a set of these tuples, the order in which they appear is completely irrelevant. A relation instance containing tuples $\{A, B, C\}$ is mathematically identical to an instance containing $\{C, A, B\}$. Database engines do not guarantee a specific order of rows unless an explicit sort command is issued.

Degree & Cardinality of a Relation

In a relational database, **degree** is the number of attributes (columns) in a relation, whereas **cardinality** is the number of tuples (rows) stored in it. For example, if a table has four columns — ID, Name, Age, and Dept — its **degree** is 4. If the table contains 100 records, then its **cardinality** is 100.

3.2 Attribute Domains, Atomicity & NULL values

Domain

For every attribute, there is a set of permitted values called its **Domain**. For instance, the domain of the `Salary` attribute is the set of all non-negative currency values.

Atomic Domains

A critical requirement of the relational model is that domains must be **Atomic**.

- **Definition:** A domain is atomic if its elements are considered indivisible units.
- **Non-Atomic Example:** If a `phone_number` attribute stores a *set* of three numbers, it is non-atomic because it has subparts (the individual numbers).
- **Atomic Example:** If we treat a phone number "555-0102" as a single string that we never split, it is atomic. If we attempt to query based on just the "Area Code" sub-part, we are treating it as non-atomic.

NULL values

When a value for an attribute is unknown or simply does not exist for a specific tuple, we use a special marker called **Null**.

1. **Unknown:** The value exists (e.g., an instructor has a phone), but we do not know it.
2. **Does Not Exist:** The value is not applicable (e.g., an instructor does not own a phone).

Note: Null values complicate database operations and should be minimized through proper design.

3.3 Database Schema

Concept

In database systems, we must distinguish between the **Database Schema**, which is the logical design or structural blueprint of the database, and the **Database Instance**, which is a snapshot of the actual data at a specific moment in time.

To use a programming analogy:

- A **Relation Schema** is equivalent to a **Type Definition** (like a `struct` or `class`). It defines the attributes and their domains but contains no data.
- A **Relation Instance** is equivalent to the **Value of a Variable**. Just as a variable's value changes as a program executes, the instance changes as data is inserted or deleted.

The **Schema** of a relation generally remains static, while the **Instance** is dynamic.

Example 28:

Consider the **Department** relation. Its **Schema** is defined as:

`department (dept_name, building, budget)`

An **Instance** of this relation might look like the following:

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Physics	Watson	70000

A crucial aspect of relational design is the use of **overlapping attributes**. Notice that `dept_name` appears in both the **Instructor** schema and the **Department** schema. This duplication is intentional; it allows us to **link data** across distinct relations. For instance, to find instructors working in the 'Watson' building, we first identify departments located in 'Watson' via the **Department** table, then match those department names in the **Instructor** table.
This pattern extends to complex university operations. A **Course** may be offered multiple times (different semesters or years). We represent these specific offerings as a **Section**. The **Section Schema** is:

`section (course_id, sec_id, semester, year, building, room_number, time_slot_id)`

To connect faculty to these sections, we use the **Teaches** relation. An **Instance** of **Teaches** acts as a bridge, mapping an `ID` to a specific class section:

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
22222	PHY-101	1	Fall	2017

By combining various schemas—such as `student`, `advisor`, `takes`, and `classroom`—a university builds a cohesive relational web.

3.4 Keys

Key

A *key*, in Database Management Systems, is either a single attribute (or column) or a set of attributes that can uniquely identify rows (or tuples) in a table. We also use keys to set up relations amongst various columns and tables of a relational database.

Types of Keys

1. Composite Key
2. Candidate Key
3. Primary Key
4. Unique Key
5. Super Key
6. Foreign Key
7. Alternate Key

Composite Key

A Composite Key is a key formed using two or more attributes together to uniquely identify a tuple in a relation. No single attribute alone can uniquely identify the tuple.

Example 29:

ENROLL Table:

StudentID	CourseID	Grade
101	DBMS	A
101	OS	B
102	DBMS	A
103	AI	B

Here StudentID is repeated and CourseID is also repeated. But the combination (StudentID, CourseID) is unique for every row. Hence (StudentID, CourseID) is the Composite Key.

Candidate Key

A Candidate Key is a **minimal** set of attributes that can uniquely identify each tuple in a relation. A relation can have multiple candidate keys.

Example 30:

STUDENT Table:

RollNo	Email	AadhaarNo	Name
1	a@x.com	5551	Amit
2	b@x.com	5552	Neha
3	c@x.com	5553	Ravi
4	d@x.com	5554	Isha

RollNo, Email, and AadhaarNo are each unique. Each of them can uniquely identify tuples. So all three are Candidate Keys.

Primary Key

A Primary Key is one candidate key selected to uniquely identify tuples in a relation. It must be unique and cannot be NULL.

Example 31:

STUDENT Table (Primary Key = RollNo):

RollNo	Name	City
1	Amit	Delhi
2	Neha	Mumbai
3	Ravi	Pune
4	Isha	Jaipur

RollNo values are unique and not NULL. Hence RollNo is chosen as the Primary Key.

Unique Key

A Unique Key is a column or a set of columns that ensures all values in a column are distinct. Unlike a Primary Key, a Unique Key **can accept a NULL value** (depending on the database system, usually one NULL is allowed), but it still prevents duplicate entries for non-null data.

Example 32:

EMPLOYEE Table (Unique Key = EmailID):

EmpID	Name	EmailID
101	Rahul	rahul@abc.com
102	Priya	priya@xyz.com
103	Vikram	NULL
104	Aman	rahul@abc.com (Error)

The EmailID column must be unique. Since **rahul@abc.com** is already assigned to EmpID 101, the system will reject the attempt to assign the same email to EmpID 104.

Super Key

A Super Key is any set of attributes that can uniquely identify a tuple. It may contain extra attributes along with a candidate key.

Example 33:

STUDENT Table:

RollNo	Email	Name
1	a@x.com	Amit
2	b@x.com	Neha
3	c@x.com	Ravi
4	d@x.com	Isha

RollNo is unique $\rightarrow$ {RollNo} is a Super Key. (RollNo, Name) is also unique $\rightarrow$ another Super Key. (RollNo, Email) is also unique $\rightarrow$ another Super Key.
Super keys can include extra attributes.

Foreign Key

A Foreign Key is an attribute in one table that refers to the Primary Key of another table. It maintains referential integrity.

Example 34:

DEPARTMENT Table (Primary Key = DNo):

DNo	DName
10	HR
20	IT
30	Sales
40	Finance

EMPLOYEE Table (Foreign Key = DNo):

EID	Name	DNo
1	Amit	10
2	Neha	20
3	Ravi	20
4	Isha	30

DNo in EMPLOYEE refers to DNo in DEPARTMENT. So DNo in EMPLOYEE is a Foreign Key.

Alternate Key

An Alternate Key is a candidate key that is not selected as the primary key.

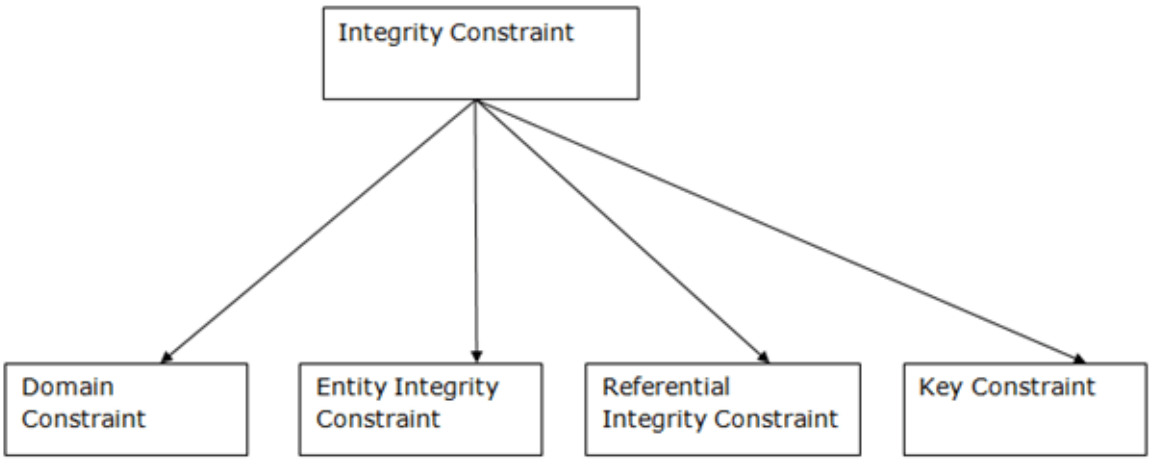
Example 35:

STUDENT Table:

RollNo	Email	AadhaarNo	Name
1	a@x.com	5551	Amit
2	b@x.com	5552	Neha
3	c@x.com	5553	Ravi
4	d@x.com	5554	Isha

Candidate Keys = RollNo, Email, AadhaarNo. If RollNo is selected as Primary Key, then Email and AadhaarNo become Alternate Keys.

3.5 Integrity Constraints



Concept

In modeling the design of the relational database we can put some restrictions like what values are allowed to be inserted in the relation, and what kind of modifications and deletions are allowed in the relation. These are the restrictions we impose on the relational database.

Domain Constraint

Every domain must contain atomic values(smallest indivisible units) which means composite and multi-valued attributes are not allowed.
We perform a datatype check here, which means when we assign a data type to a column we limit the values that it can contain.

Example 36:

If we assign the datatype of attribute age as int, we can't give it values other than int datatype.

EID	Name	Phone
01	Bikash Dutta	123456789 234456678

Employee Information

In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

Key Constraints or Uniqueness Constraints

These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
A relation can have multiple keys or candidate keys (minimal superkey), out of which we choose one of the keys as the primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.
Null values are not allowed in the primary key, hence Not Null constraint is also part of the key constraint.

Example 37:

EID	Name	Phone
01	Bikash	6000000009
02	Paul	9000090009
01	Tuhin	9234567892

Employee Information

In the above table, EID is the primary key, and the first and the last tuple have the same value in EID ie 01, so it is violating the key constraint.

Entity Integrity Constraints

Entity Integrity constraints say that no primary key can take a NULL value, since using the primary key we identify each tuple uniquely in a relation.

Example 38:

EID	Name	Phone
01	Bikash	9000900099
02	Paul	600000009
NULL	Sony	9234567892

Employee Information

In the above relation, EID is made the primary key, and the primary key can't take NULL values but in the third tuple, the primary key is null, so it is violating Entity Integrity constraints.

Referential Integrity Constraints

The Referential integrity constraint is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
This constraint is enforced through a foreign key, when an attribute in the foreign key of relation R1 has the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.
The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some tuple in relation R2, or can take NULL values, but can't be empty.

Example 39:

EID	Name	DNO
01	Divine	12
02	Dino	22
04	Vivian	14

Employee Information

DNO	Place
12	Jaipur
13	Mumbai
14	Delhi

Department Information

In the above tables, the DNO of Table **Employee Information** is the foreign key, and DNO in Table **Department Information** is the primary key. DNO = 22 in the foreign key of **Employee Information** table is not allowed because DNO = 22 is not defined in the primary key of table **Department Information**. Therefore, Referential integrity constraints are violated here.

Handling Violations: ON DELETE SET NULL

The **ON DELETE SET NULL** strategy is used when a record in the referenced (parent) relation is deleted. To maintain referential integrity, the system automatically sets the foreign key values in the referencing (child) relation to `NULL` for all tuples that were linked to the deleted record.
This approach is typically used when the child record should continue to exist even if its association with the parent is removed.

Example 40:

Suppose we delete the record for **DNO = 14** (Delhi) from the Department table. With *on-delete set null* enabled, the **Employee Information** table is updated as follows:

DNO	Place
12	Jaipur
13	Mumbai

Department Information (After Deletion of DNO 14)

EID	Name	DNO
01	Divine	12
02	Dino	13
04	Vivian	**NULL**

Employee Information (DNO for Vivian set to NULL)

Here, Vivian is still a valid employee in the database, but she is currently not assigned to any department since DNO 14 no longer exists.

Handling Violations: ON UPDATE CASCADE

The **ON UPDATE CASCADE** strategy ensures that any modification made to a Primary Key value in the referenced relation is automatically "cascaded" to all corresponding Foreign Key values in the referencing relation. This maintains the logical link between tables without requiring manual updates to multiple records across the database.

Example 41:

If the **DNO** for the *Jaipur* branch is changed from **12** to **50** in the Department table, the **Employee Information** table will reflect this change automatically:

DNO	Place
50	Jaipur
13	Mumbai
14	Delhi

Department Information (Primary Key Updated)

EID	Name	DNO
01	Divine	**50**
02	Dino	13
04	Vivian	14

Employee Information (Foreign Key Cascaded)

The **DNO** for Divine was automatically updated from 12 to 50, ensuring the referential integrity constraint is preserved.

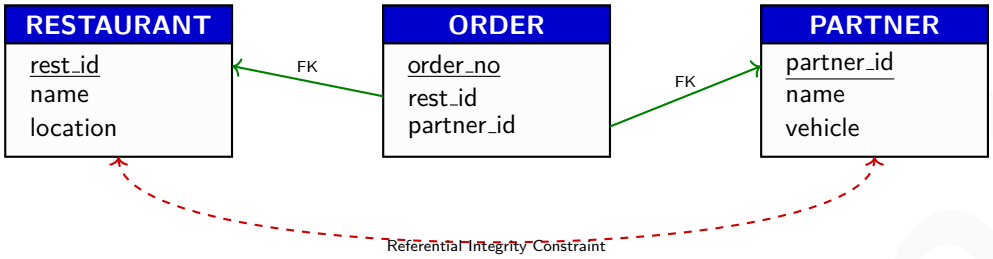
3.6 Schema Diagrams

In a schema diagram, each table is depicted as a box. The **Relation Name** sits at the top, and the **Primary Key**—the unique identifier for every record—is underlined.

Consider a simple Food Delivery database with three relations:

- **Restaurant**: Stores details of eateries like "Haldiram's" or "Empire Restaurant".

- **Partner**: Details of the person delivering the order.
- **Order**: The bridge that connects a customer, a restaurant, and a delivery partner.
- **Primary Key:** The unique field (underlined), e.g., rest_id.
- **Foreign Key (FK):** A link (green arrow) ensuring that a restaurant ID in the Order table actually exists in the Restaurant table.
- **Referential Integrity:** Ensures that the overall data across different tables (like Restaurant and Partner) remains consistent.



3.7 Reducing E-R Diagrams to Relational Schema

Concept

Both the Entity–Relationship (E–R) model and the relational database model provide abstract, logical frameworks for modeling real-world enterprises at the schema level. Because these models are based on compatible structural and semantic design principles, an E–R schema can be formally mapped into an equivalent relational schema. Under this mapping, each entity set and each relationship set in the E–R design is transformed into a separate relation schema, with the relation name conventionally derived from the corresponding entity or relationship set.

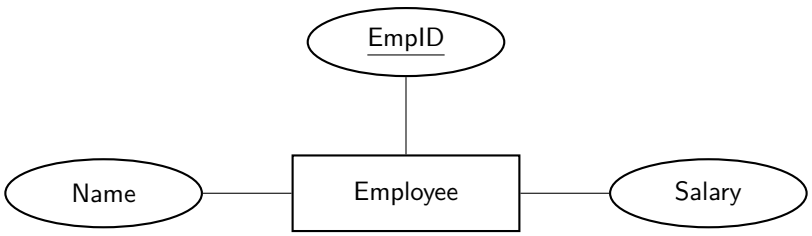
3.7.1 Rules for Reducing E-R Diagrams to Relational Schema

Rule 1: Strong Entity Set with Simple Attributes

- A strong entity set has its own primary key and does not depend on any other entity set for identification.
- If all attributes are simple and single-valued, the entity set maps to exactly one relational table.
- Each attribute of the entity set becomes one column in the table.
- The key attribute(s) of the entity set become the primary key of the table.
- If the ER key is composite, the relational primary key is also composite.
- Only stored attributes are mapped — derived attributes are not included in the table.

Example 42: Strong Entity Set with Simple Attributes — Employee

ER Diagram



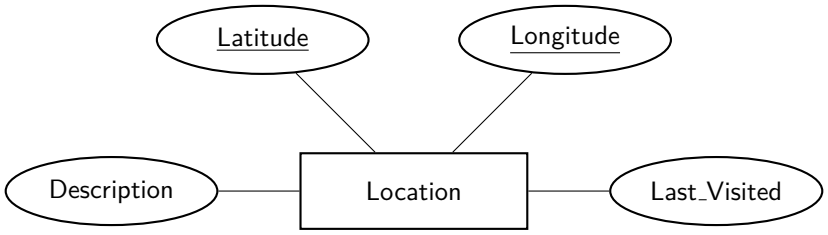
Relational Mapping

$$Employee(\underline{EmpID},\ Name,\ Salary)$$

Conversion rule used: One strong entity with simple attributes → one table; key becomes primary key.

Example 43: Strong Entity Set with Composite Key — Location

ER Diagram



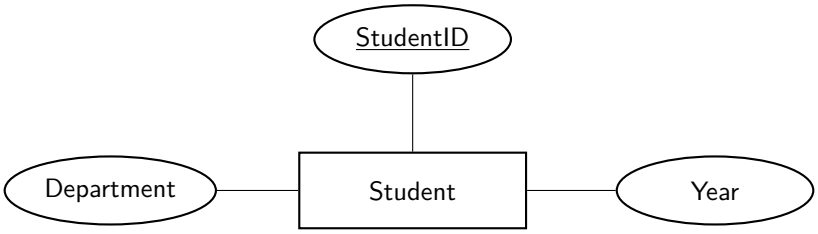
Relational Mapping

$$Location(\underline{Latitude, Longitude},\ Description,\ Last\_Visited)$$

Conversion rule used: Composite key in ER → composite primary key in relation.

Example 44: Strong Entity Set — Student

ER Diagram



Relational Mapping with Arrow View

$$\text{Student entity} \implies Student(\underline{StudentID},\ Department,\ Year)$$

Conversion rule used: All simple attributes map directly to columns.

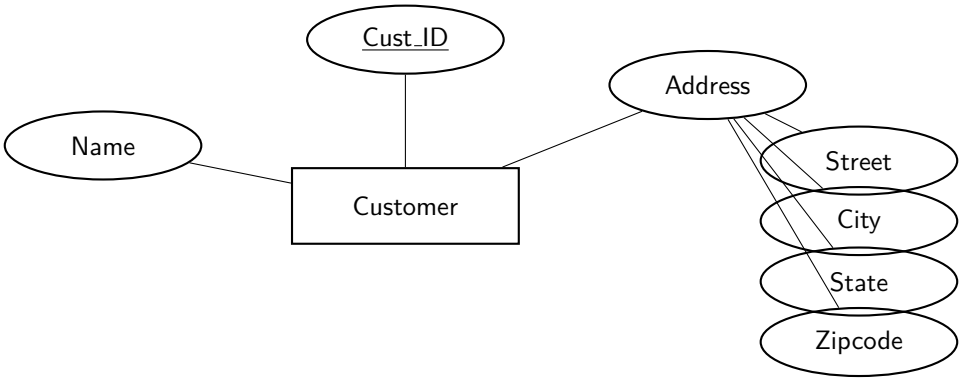
Rule 2: Strong Entity Set with Composite Attributes

- A strong entity set may contain composite attributes (attributes that can be divided into sub-attributes).
- Even if composite attributes exist, only one relational table is created for the entity set.
- During ER → relational mapping, the composite attribute itself is not stored as a column.
- Only the simple (atomic) components of each composite attribute are included as table columns.
- The primary key of the entity set remains the primary key of the table.

- This rule applies only when composite attributes are single-valued (not multivalued).
- If a composite attribute contains derived parts, derived components are not stored.

Example 45: Strong Entity with Composite Attribute — Customer Address

ER Diagram



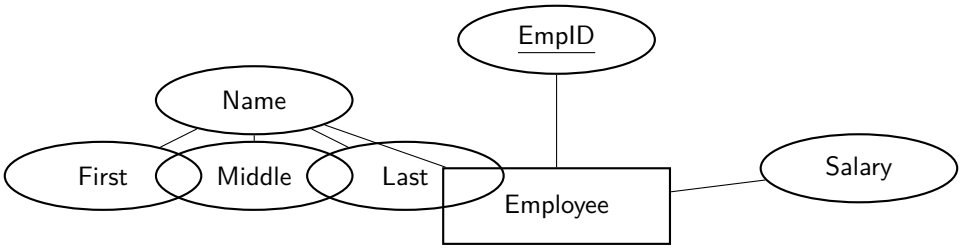
Relational Mapping

Customer entity with composite Address $\implies$ $Customer(\underline{cust\_id},\ name,\ street,\ city,\ state,\ zipcode)$

Rule used: Composite attribute → store only its simple components.

Example 46: Strong Entity with Composite Attribute — Employee Name

ER Diagram



Relational Mapping

Employee with composite Name $\implies$ $Employee(\underline{EmpID},\ first,\ middle,\ last,\ salary)$

Rule used: Composite attribute is flattened into atomic columns.

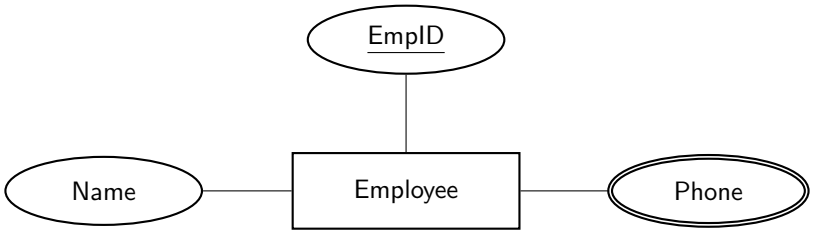
Rule 3: Strong Entity Set with Multi-Valued Attributes

- A multi-valued attribute can store multiple values for a single entity instance.
- A strong entity set having one or more multi-valued attributes cannot be mapped to a single table without redundancy.
- One main table is created for the entity set containing all simple, single-valued attributes and the primary key.

- For each multi-valued attribute, a separate relation (table) is created.
- The new relation contains:
  - the multi-valued attribute (atomic form only),
  - the primary key of the owner entity set.
- The primary key of the new relation is a composite key: (primary key of entity set + multi-valued attribute value).
- The primary key of the entity set appears as a foreign key in the new relation.
- If there are multiple multi-valued attributes, create one separate table per attribute.
- If a multi-valued attribute is composite, include all its simple components in the new table.

Example 47: Strong Entity with Multi-Valued Attribute — Employee Phone Numbers

ER Diagram



Relational Mapping

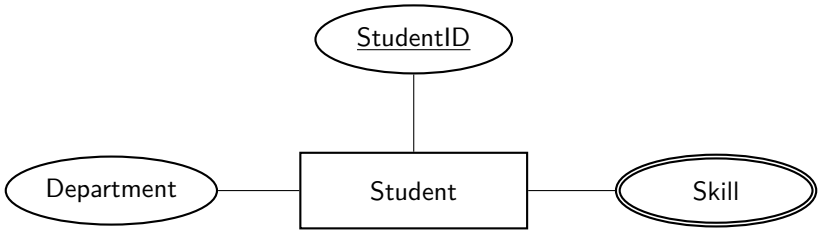
$$\text{Employee entity} \implies Employee(\underline{EmpID},\ Name)$$

$$\text{Multi-valued attribute Phone} \implies Employee\_Phone(\underline{EmpID, Phone})$$

Rule used: Multi-valued attribute → separate table with (PK + attribute) as composite key.

Example 48: Strong Entity with Multi-Valued Attribute — Student Skills

ER Diagram



Relational Mapping

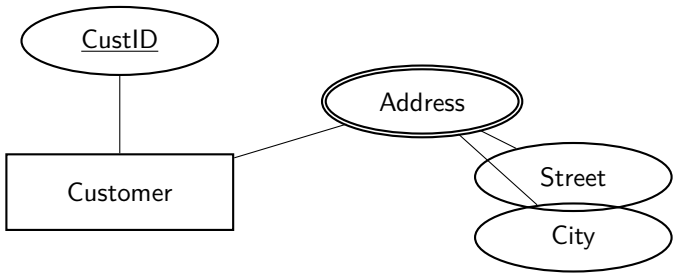
$$Student(\underline{StudentID},\ Department)$$

$$Student\_Skill(\underline{StudentID, Skill})$$

Rule used: Each multi-valued attribute gets its own relation.

Example 49: Composite Multi-Valued Attribute — Customer Addresses

ER Diagram



Relational Mapping

$$Customer(\underline{CustID})$$

$$Customer\_Address(\underline{CustID, Street, City})$$

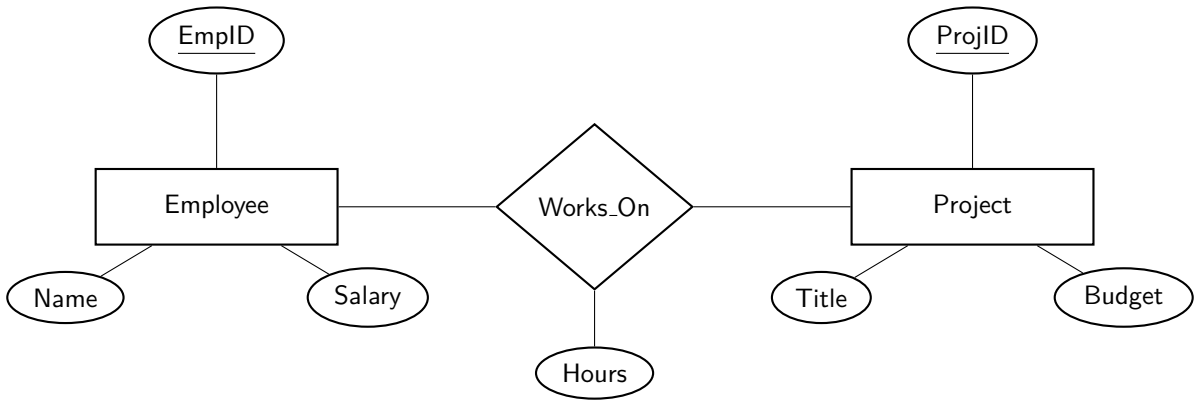
Rule used: Composite multi-valued attribute $\rightarrow$ new table with all simple components + entity PK.

Rule 4: Relationship Set into a Table

- A relationship set can be mapped to a separate relational table.
- One table is created for the relationship set itself.
- The columns of the relationship table include:
  - Primary key attributes of all participating entity sets
  - Descriptive attributes of the relationship set (if any)
- Primary key attributes of participating entities appear as foreign keys in the relationship table.
- The relationship table stores only key attributes from entities — not their non-key attributes.
- The primary key of the relationship table is formed from the set of participating entity primary keys (combined key).
- Descriptive attributes are included in the table but are not part of the primary key unless explicitly specified.
- This rule describes only the basic relationship-to-table mapping; cardinality-based optimizations are handled separately.

Example 50: Relationship Set into a Table — Works_On

ER Diagram



Relational Mapping

$$Employee(\underline{EmpID}, Name, Salary)$$

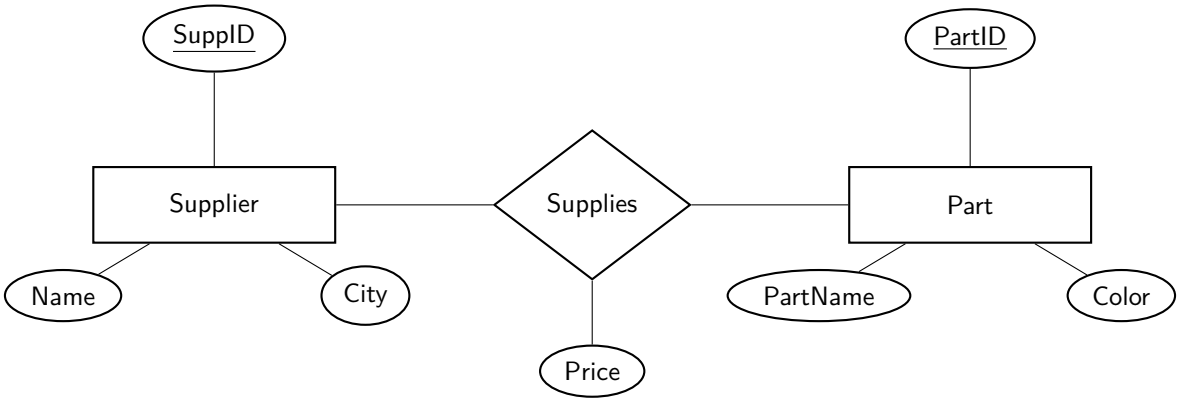
$$Project(\underline{ProjID}, Title, Budget)$$

$$\text{Works\_On relationship} \implies Works\_On(\underline{EmpID}, \underline{ProjID}, Hours)$$

Rule used: Relationship table contains only participating PKs + relationship attributes.

Example 51: Relationship Set into a Table — Supplies

ER Diagram



Relational Mapping

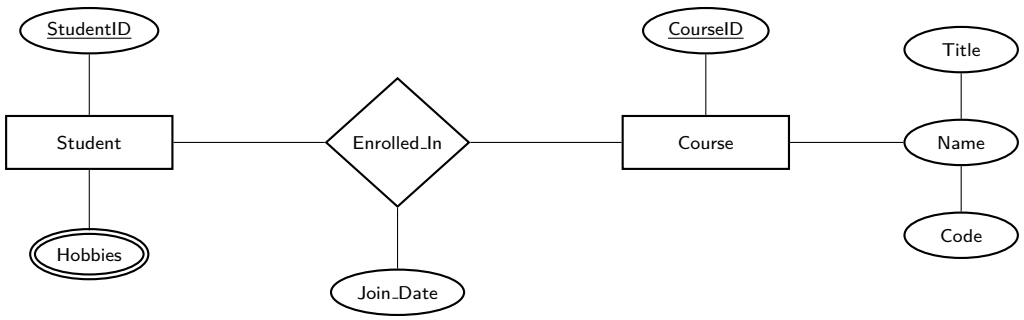
$$Supplier(\underline{SuppID}, Name, City)$$

$$Part(\underline{PartID}, PartName, Color)$$

$$Supplies(\underline{SuppID}, \underline{PartID}, Price)$$

Rule used: Only entity keys propagate into the relationship table, not other attributes.

Example 52: Rule 1 -4 Combined Example



Relational Mapping

$$Student(\underline{StudentID})$$

$$Student\_Hobbies(\underline{StudentID, Hobby})$$

$$Course(\underline{CourseID}, Title, Code)$$

$$Enrollment(\underline{StudentID, CourseID}, Join\_Date)$$

Conversion Summary:

- Multivalued → separate relation.
- Composite → decomposed.
- Descriptive → stored in relationship.

Mapping Principles

- **Cardinality:** Determines **FK placement** or if a **new table** is needed.
- **Participation:** Determines **nullability** (*NULL* vs. *NOT NULL*).
- **Descriptive Attributes**: Follow the FK.
- **Goal:** Minimize tables to reduce join overhead.

Concept	Mapping Impact	Constraint
Cardinality	Table/FK Location	**UNIQUE** (for 1:1)
Partial	Optional Relationship	**FK NULL** allowed
Total	Mandatory Relationship	**FK NOT NULL**

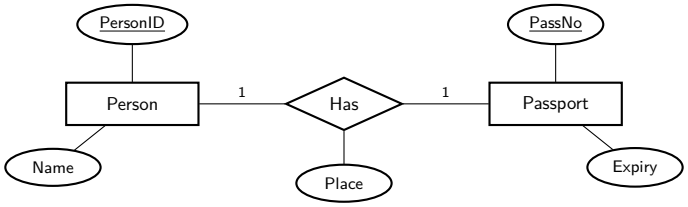
Rule 5: Mapping 1:1 Relationships

- Separate tables are created for each entity; a relationship table is **not required**.
- The PK of one entity is added as a **Foreign Key (FK)** in the other.
- The FK column must be **UNIQUE** to maintain the 1:1 ratio.
- Store relationship attributes in the table receiving the Foreign Key.

Participation Mapping Strategy

Participation	FK Location	FK Constraint
Both sides Partial	Either side (Convenience)	**NULL** allowed
One side Total	Mandatory (Total) side	**NOT NULL**
Both sides Total	**Merge Tables**	**NOT NULL**

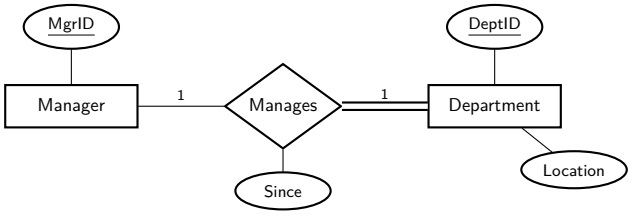
Example 53: Example 1: 1:1 Partial Participation



Relational Tables:

- **Person** (PersonID, Name)
- **Passport** (PassNo, Expiry, PersonID, Place)
  (PersonID is a Foreign Key. We merge the relationship into the side that is most likely to have a passport to avoid nulls.)

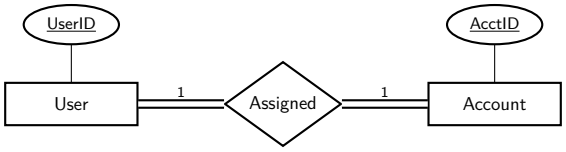
Example 54: Example 2: 1:1 Total Participation on one side



Relational Tables:

- **Manager** (MgrID)
- **Department** (DeptID, Location, MgrID, Since)
  (MgrID is a Foreign Key. Foreign Key is placed in the 'Total Participation' side to ensure every Department has a Manager.)

Example 55: Example 3: 1:1 Total Participation Both Sides



Relational Tables:

- **User_Account** (UserID, AcctID)
  (Since participation is total on both sides, the two entities can be merged into a single table to improve efficiency and eliminate joins.)

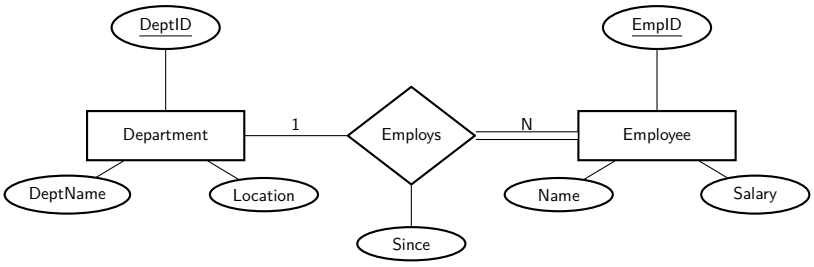
Rule 6: Mapping 1:N Relationships

- The PK of the **1-side** is added as a **Foreign Key (FK)** in the **N-side** relation.
- This direction is **mandatory** to maintain functional dependency.
- Relationship attributes are stored in the **N-side** relation.

Participation	FK Location	FK Constraint
N-side Partial	N-side (Mandatory)	**NULL** allowed
N-side Total	N-side (Mandatory)	**NOT NULL**
Both sides Total	N-side (Mandatory)	**NOT NULL***

**Note: Total participation on the 1-side requires additional triggers/assertions.*

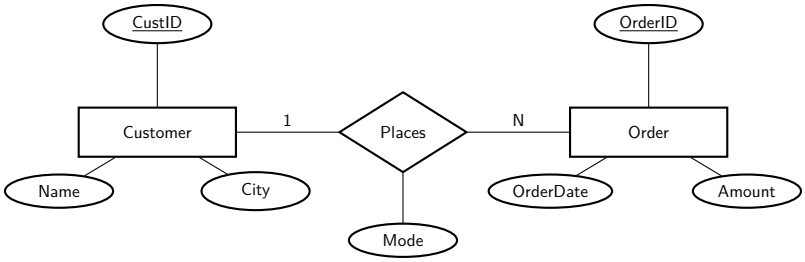
Example 56: Example 4: 1:N Total Participation on N-side



Relational Tables:

- **Department** (DeptID, DeptName, Location)
- **Employee** (EmpID, Name, Salary, DeptID, Since)
  (The Primary Key of the 1-side (DeptID) is added to the N-side (Employee) as a Foreign Key. Relationship attributes like 'Since' also move to the N-side table.)

Example 57: Example 5: 1:N Partial Participation



Relational Tables:

- **Customer** (CustID, Name, City)
- **Order** (OrderID, OrderDate, Amount, CustID, Mode)
  (Even with partial participation, the standard 1:N rule applies: the Foreign Key goes to the N-side. If an Order doesn't have a Customer, the CustID field will simply be NULL.)

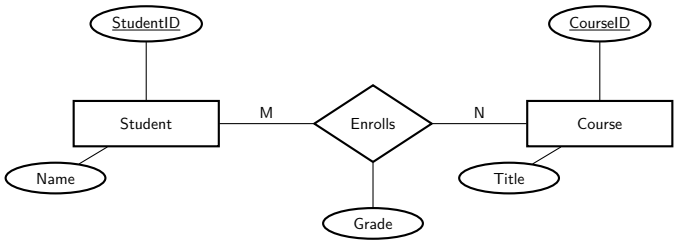
Rule 7: Mapping M:N Relationships

- A **separate relationship table** is mandatory to avoid data redundancy.
- The table contains the PKs of both entities as **Foreign Keys (FKs)**.
- The **Primary Key** is a composite key of both participating FKs.
- Relationship attributes are stored exclusively in this new table.

Participation	FK Location	FK Constraint
Both sides Partial	New Separate Table	**NOT NULL***
One side Total	New Separate Table	**NOT NULL***
Both sides Total	New Separate Table	**NOT NULL***

**Note: In an M:N bridge table, FKs are always NOT NULL. Total participation is enforced by ensuring every record in the entity table has at least one entry in this relationship table via application logic.*

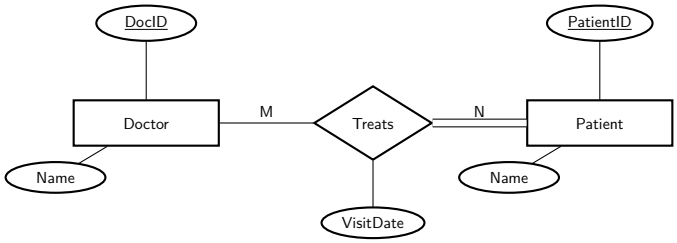
Example 58: Rule 7: M:N Relationship (Both Sides Partial)



Relational Tables:

- **Student** (StudentID, Name)
- **Course** (CourseID, Title)
- **Enrolls** (StudentID, CourseID, Grade)
  (A new table is created. The Primary Key is the combination of both Foreign Keys.)

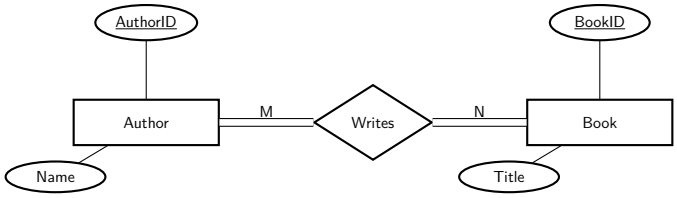
Example 59: Rule 7: M:N Relationship (One Side Total)



Relational Tables:

- **Doctor** (DocID, Name)
- **Patient** (PatientID, Name)
- **Treats** (DocID, PatientID, VisitDate)
  (Total participation on the Patient side means every PatientID must appear at least once in the 'Treats' table.)

Example 60: Rule 7: M:N Relationship (Both Sides Total)



Relational Tables:

- **Author** (AuthorID, Name)
- **Book** (BookID, Title)
- **Writes** (AuthorID, BookID)
  (Total participation on both sides is enforced at the application/database constraint level, ensuring no ID exists in Author or Book without a corresponding entry in Writes.)

Rule 8: Recursive (Unary) Relationships

A recursive (unary) relationship occurs when an entity set is related to itself.

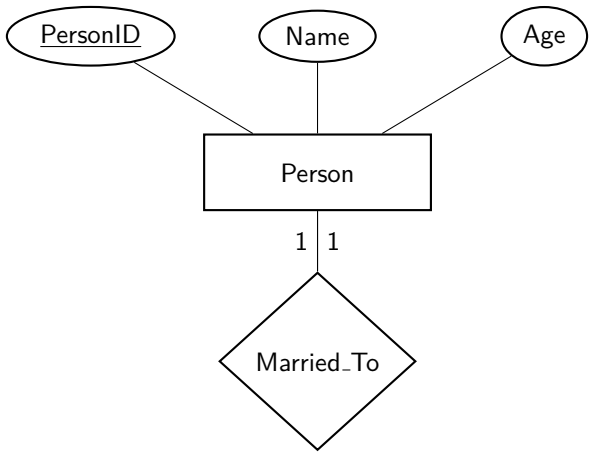
- Only one entity set participates — but with different roles.
- Mapping depends on the cardinality ratio.
- Role names must be used to distinguish participation.

Mapping summary

- Recursive 1:1 → Add a self-referencing foreign key in the entity table (or separate relation if relationship has attributes).
- Recursive 1:N → Add a self-referencing foreign key on the N-side (same table).
- Recursive M:N → Create a separate relationship table with two foreign keys referencing the same entity table.

Example 61: Recursive 1:1 — Person Married_To Person

ER Diagram



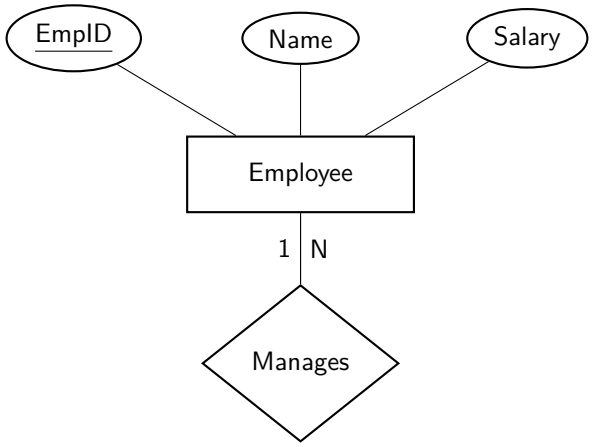
Relational Mapping

$$Person(\underline{PersonID}, Name, Age, SpouseID)$$

SpouseID → FK referencing Person(PersonID), UNIQUE.
Rule used: Recursive 1:1 → self foreign key.

Example 62: Recursive 1:N — Employee Manages Employee

ER Diagram



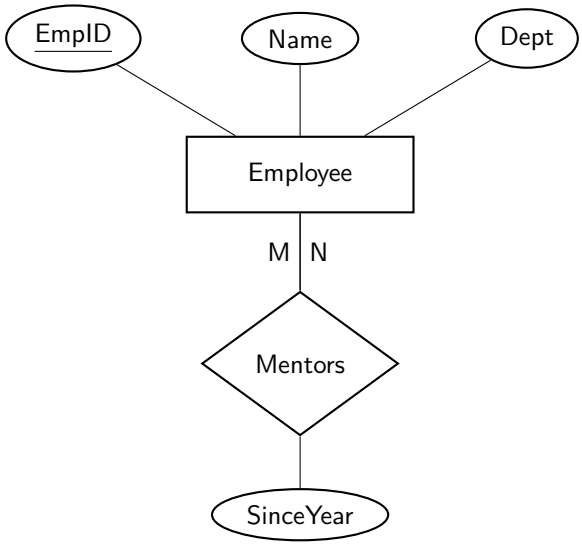
Relational Mapping

$$Employee(\underline{EmpID}, Name, Salary, ManagerID)$$

ManagerID → FK referencing Employee(EmpID).
Rule used: Recursive 1:N → FK stored on N-side (same table).

Example 63: Recursive M:N — Employee Mentors Employee

ER Diagram



Relational Mapping

$$Employee(\underline{EmpID}, Name, Dept)$$

$$Mentors(\underline{MentorID, MenteeID}, SinceYear)$$

MentorID → FK to Employee(EmpID) MenteeID → FK to Employee(EmpID)
Rule used: Recursive M:N → separate relation with two FKs to same table.

Rule 9: Binary Relationship with Weak Entity

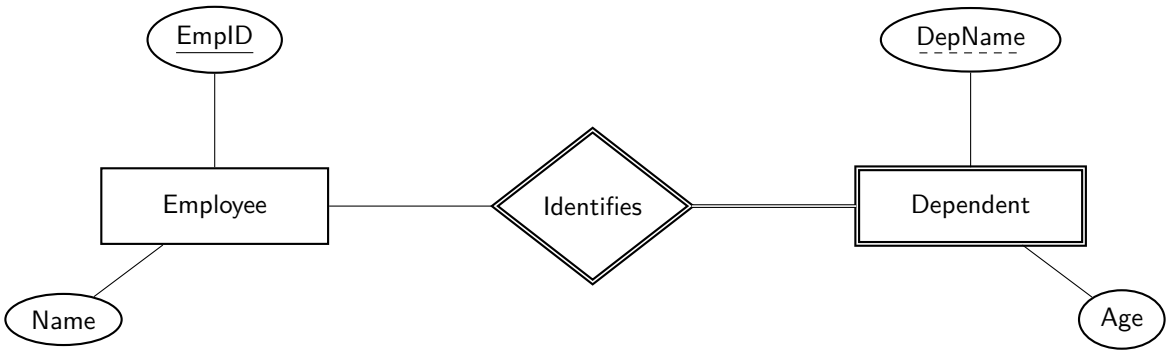
- A weak entity set does not have a full primary key of its own.
- It depends on an identifying (owner) entity through an identifying relationship.
- Weak entity has:
  - Owner entity key
  - Partial key (discriminator)
- Participation of weak entity in identifying relationship is always total.
- Identifying relationship is not mapped as a separate table (unless it has its own attributes).

Mapping procedure
For each weak entity set W with owner entity E:

- Create a relation for the owner entity E as usual.
- Create a relation for weak entity W.
- Include all simple attributes of W.
- Add the primary key of owner entity E into W as a foreign key.
- Primary key of weak entity table = (Owner primary key + Weak entity partial key).
- Add foreign key constraint from weak table to owner table.

Example 64: Weak Entity — Employee and Dependent

Each Employee can have many Dependents. Dependent is a weak entity identified by Employee + DepName.
ER Diagram



Relational Mapping

$$Employee(\underline{EmpID}, Name)$$

$$Dependent(\underline{EmpID, DepName}, Age)$$

EmpID → FK referencing Employee(EmpID)
Primary Key Rule: Owner PK + Partial Key

3.7.2 Converting n-ary Relationships

Rule 10: Mapping n-ary Relationships

- Used when a relationship involves **three or more** entity sets (e.g., Ternary).
- **Mandatory Table:** Unlike 1:1 or 1:N, these **always** require a separate relationship table.
- **Attributes:** All relationship attributes are stored in this new table.

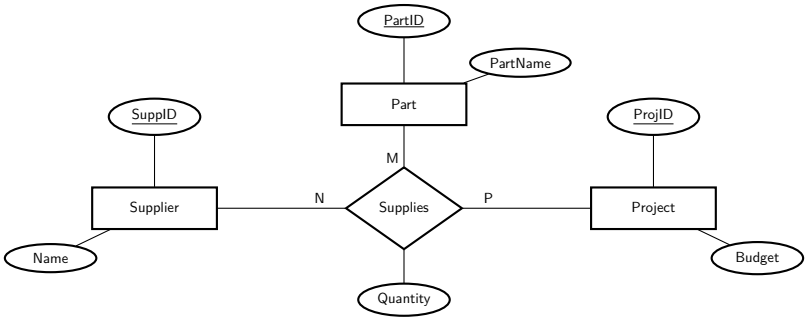
Step-by-Step Reduction:

1. Create standard tables for all participating entity sets (e.g., E_1, E_2, E_3).
2. Create a **new relationship table** (R).
3. Add the **Primary Keys** of all participating entities into table R as **Foreign Keys**.
4. Define the **Primary Key of** R as the combination (composite) of all those Foreign Keys.

Participation	FK Location	FK Constraint
n-ary Relation	New Separate Table	**NOT NULL** (Part of PK)

**Note: When an entity has a "1" constraint, it becomes functionally dependent on the others, so its key is excluded from the composite PK.*

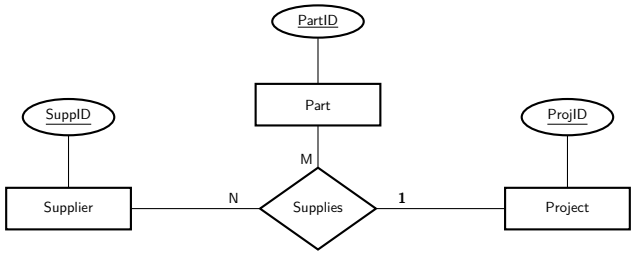
Example 65: n-ary Relationship — Supplier Supplies Part to Project



Relational Mapping (N:M:P):

- **Supplier**(SupplID, Name)
- **Part**(PartID, PartName)
- **Project**(ProjID, Budget)
- **Supplies**(SupplID, PartID, ProjID, Quantity) → *All 3 are FKs*

Example 66: Relational Mapping (1:N:M)



Relational Mapping (1:N:M):

- **Supplies**(SupplID, PartID, ProjID, Quantity)
- *The ProjectID is a Foreign Key but **not** part of the Primary Key because of the "1" constraint.*

3.7.3 Summary

Rule	Concept	Mapping Action	Key / Constraint
Rule 1	Strong Entity	New Table	Single Primary Key
Rule 2	Composite Attr.	Flatten into Entity Table	Simple columns
Rule 3	Multivalued Attr.	**New Table**	$(PK +$ Attribute$)$
Rule 5	1:1 Relationship	FK in one table	**UNIQUE**
Rule 6	1:N Relationship	FK in **N-side** table	**NOT NULL** (if Total)
Rule 7	M:N Relationship	**New Table**	Composite PK
Rule 9	Weak Entity	Owner PK in Weak Table	Composite PK
Rule 10	n-ary Relationship	**New Table**	Cardinality dependent

- **Rule 4 Note:** Basic binary relationships are mapped by establishing Foreign Key (FK) links based on the specific cardinality (Rules 5, 6, 7).
- **Participation Impact:** Only affects **Rules 5 and 6** regarding FK location and NULL/NOT NULL constraints.

General Procedure for ER-to-Relational Mapping with multiple Entity Sets and Relationship Sets

Step 1 (Rule 1, 2): Strong Entities + Flatten Composite Attributes

↓

Step 2 (Rule 9): Weak Entities (Depends on Step 1)

↓

Step 3 (Rule 3): Multivalued Attributes → New Tables

↓

Step 4 (Rule 5): Binary 1:1 Relationships

↓

Step 5 (Rule 6): Binary 1:N Relationships

↓

Step 6 (Rule 7): Binary M:N Relationships (**New Table**)

↓

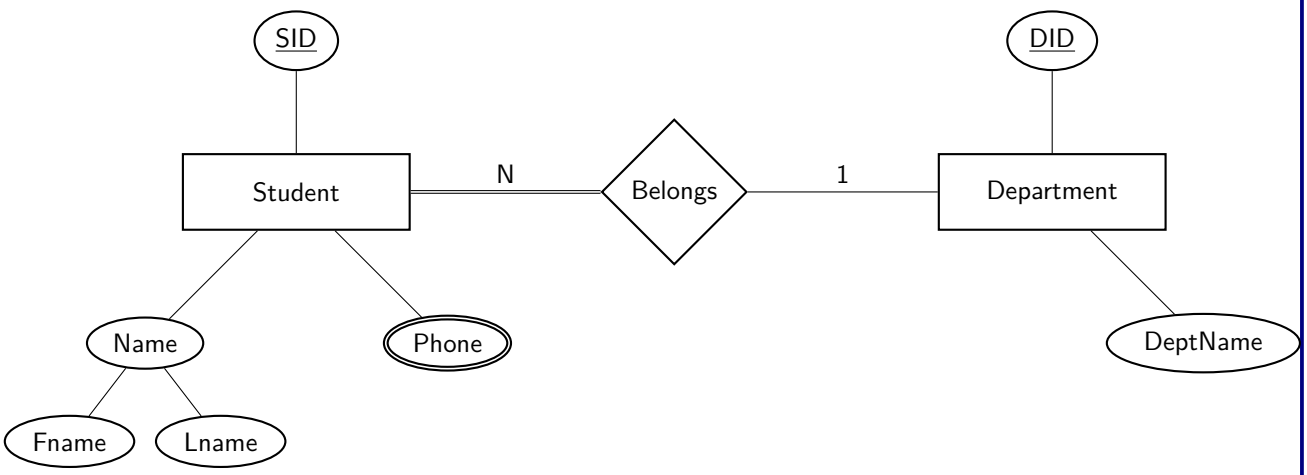
Step 7 (Rule 10): n-ary Relationships (**New Table**)

3.8 Solved Examples

Example 67: University Registration Model — Find Minimum Tables

A University stores Students and Departments.
Student has composite Name (Fname, Lname) and multivalued Phone. Each Student must belong to exactly one Department (total participation). A Department may have many Students.

ER Diagram



Find: Minimum number of tables

Solution

Student → strong entity with composite + multivalued Composite expands, multivalued separate table 1:N → FK on N side with NOT NULL

$$Department(\underline{DID}, DeptName)$$

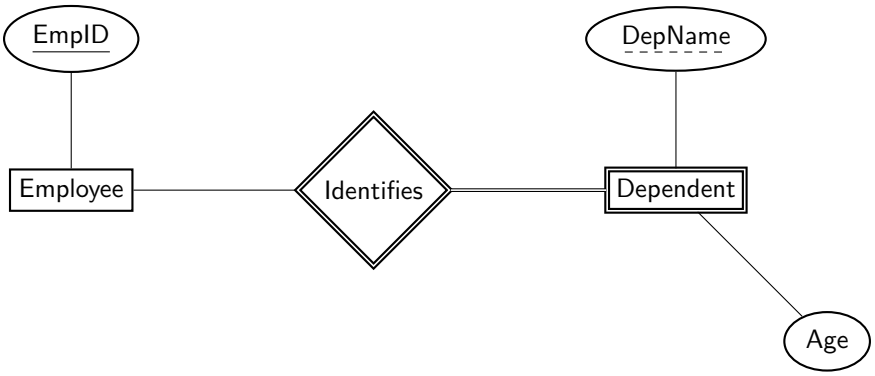
$$Student(\underline{SID}, Fname, Lname, DID \text{ NOT NULL})$$

$$StudentPhone(\underline{SID}, \underline{Phone})$$

Minimum tables = 3

Example 68: Employee–Dependent Weak Entity — Find Minimum Tables

Dependent is weak, identified by Employee and DepName. Employee has total identifying relationship.
ER Diagram



Solution
Weak entity absorbs identifying relationship.

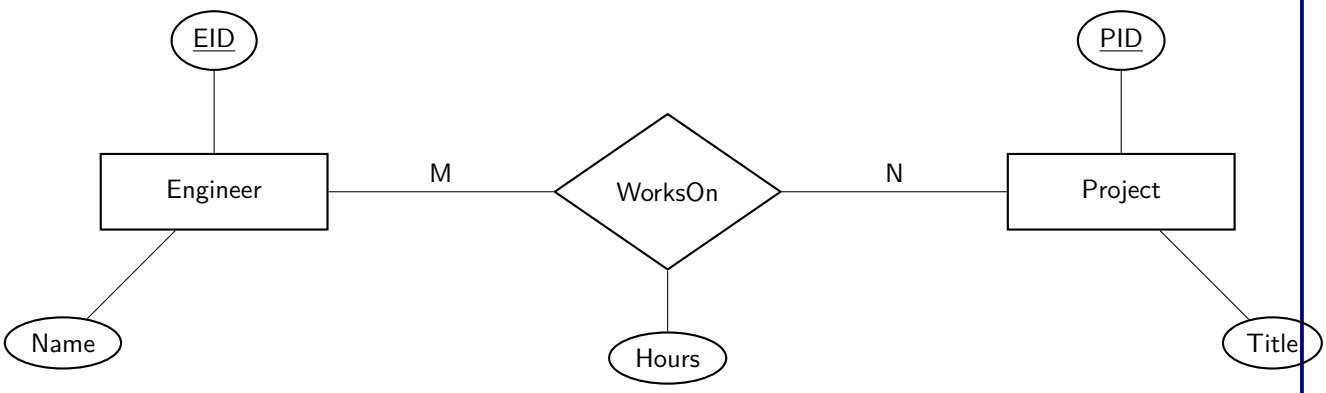
$$Employee(\underline{EmpID})$$

$$Dependent(\underline{EmpID}, \underline{DepName}, Age)$$

Minimum tables = 2

Example 69: Engineer–Project Assignment

ER Diagram



Find the minimum number of tables required after ER-to-Relational mapping.

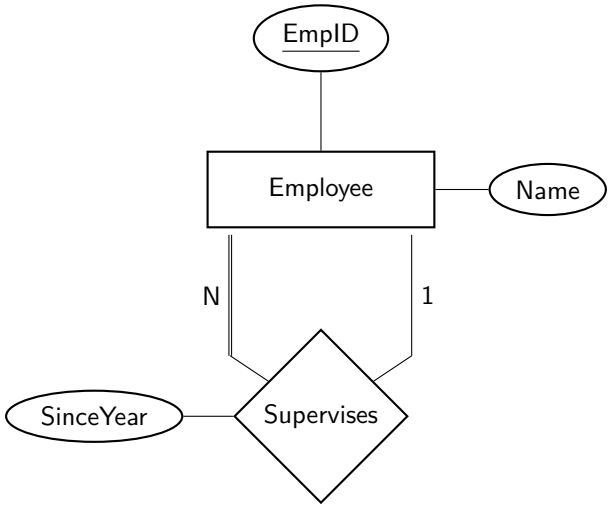
Solution
M:N relationship with descriptive attribute $\implies$ separate relation required.

$$Engineer(\underline{EID}, Name)$$
$$Project(\underline{PID}, Title)$$
$$WorksOn(\underline{EID}, \underline{PID}, Hours)$$

Minimum tables = 3

Example 70: Recursive Supervision Relationship

ER Diagram



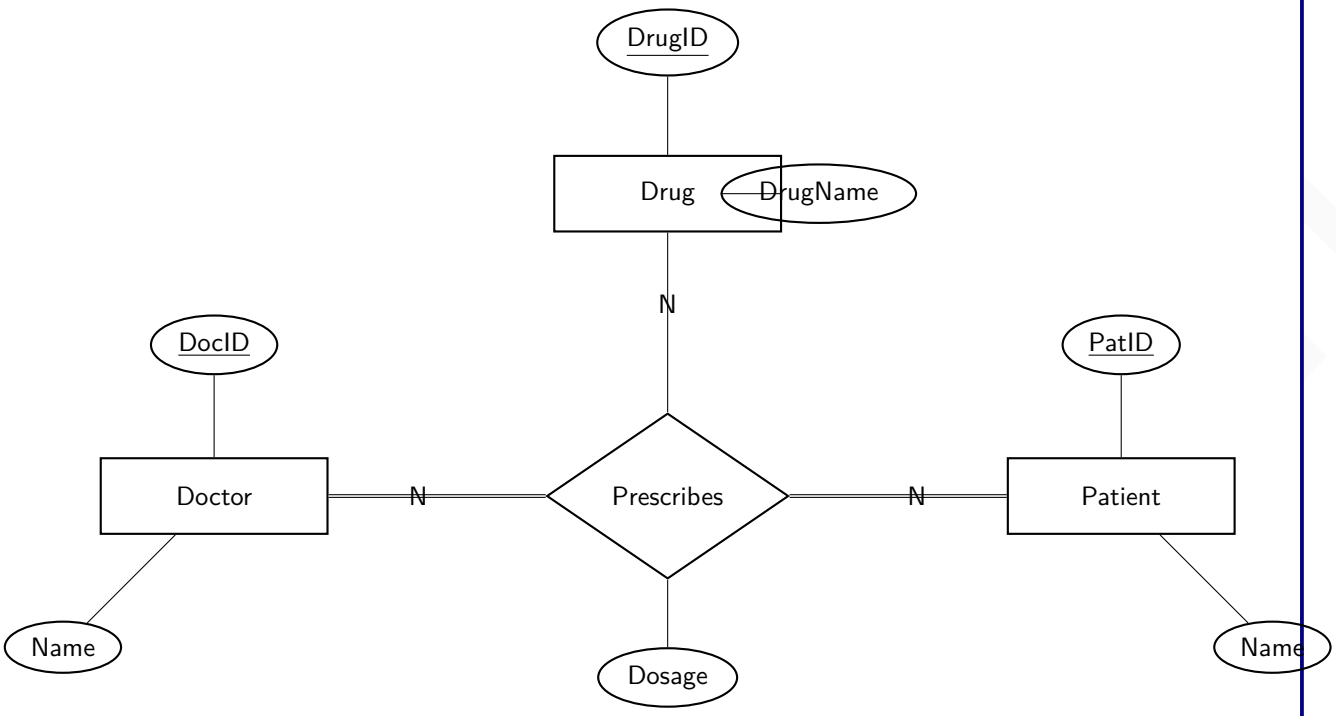
Find the minimum number of tables required after ER-to-Relational mapping.
Solution
Recursive 1:N $\implies$ no separate relation. Add self foreign key and relationship attributes to N-side.

$$Employee(\underline{EmpID}, Name, SupervisorID, SinceYear)$$

SupervisorID $\rightarrow$ FK referencing Employee.EmpID (NOT NULL due to total participation on N-side)
Minimum tables = 1

Example 71: Doctor–Patient–Drug Prescription

ER Diagram



Find the minimum number of tables required after ER-to-Relational mapping.
Solution
Ternary (n-ary) relationship $\implies$ always separate relation.

$$Doctor(\underline{DocID}, Name)$$

$$Patient(\underline{PatID}, Name)$$

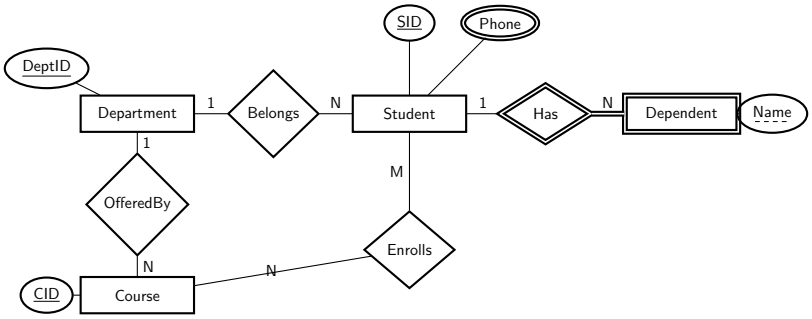
$$Drug(\underline{DrugID}, DrugName)$$

$$Prescribes(\underline{DocID, PatID, DrugID}, Dosage)$$

DocID, PatID $\rightarrow$ NOT NULL FKs (total participation)
Minimum tables = 4

Example 72: Complex Mapping: University Management System

1. ER Diagram



2. Step-by-Step Relational Reduction

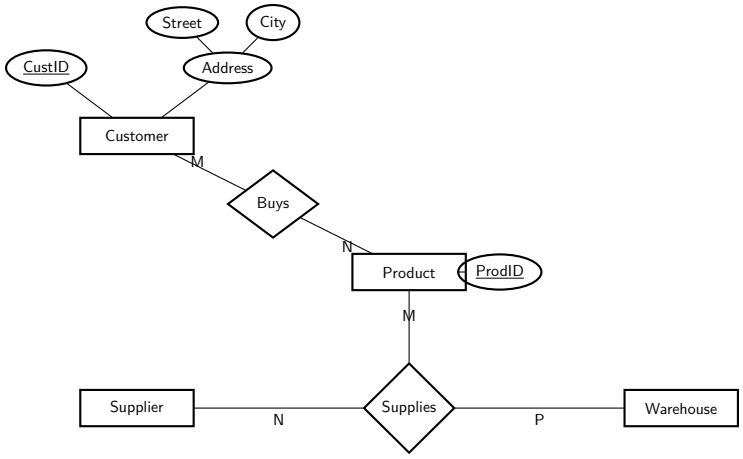
1. **Step 1 & 2 (Strong Entities):**
   - $Department(\underline{DeptID}, Name)$
   - $Student(\underline{SID}, Name, Age)$

- $Course(\underline{CID}, Title)$

2. **Step 3 (Weak Entity - Rule 9):**
   - $Dependent(\underline{\textbf{SID}, Name}, Relationship)$ *(SID is FK)*
3. **Step 4 (Multivalued - Rule 3):**
   - $Student\_Phone(\underline{\textbf{SID}, Phone})$ *(SID is FK)*
4. **Step 5 (1:N - Rule 6):**
   - Add $DeptID$ to $Student$ table: $Student(\ldots, \textbf{DeptID})$
   - Add $DeptID$ to $Course$ table: $Course(\ldots, \textbf{DeptID})$
5. **Step 6 (M:N - Rule 7):**
   - $Enrollment(\underline{\textbf{SID, CID}}, Grade)$ *(Both are FKs)*

Example 73: Supply Chain Management

1. ER Diagram

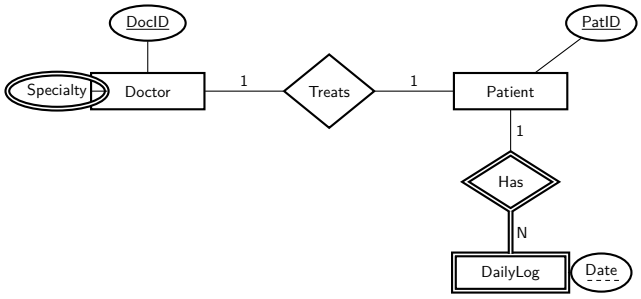


2. Relational Solution

- **Customer**(CustID, Name, **Street, City**) *(Rule 2: Flattened)*
- **Product**(ProdID, Description) *(Rule 1: Strong)*
- **Supplier**(SuppID, SName) — **Warehouse**(WhID, Location)
- **Buys**(**CustID, ProdID**, Date) *(Rule 7: Binary M:N)*
- **Supplies**(**ProdID, SuppID, WhID**, Qty) *(Rule 10: Ternary N:M:P)*

Example 74: Hospital Administration

1. ER Diagram



2. Relational Solution

- **Doctor**(<u>DocID</u>, Name)
- **Doctor_Specialty**(**<u>DocID, Specialty</u>**) *(Rule 3: Multivalued)*
- **Patient**(<u>PatID</u>, Name, **DocID**) *(Rule 5: 1:1, FK in Patient)*
- **DailyLog**(**<u>PatID, Date</u>**, Observation) *(Rule 9: Weak Entity)*

3.9 Problems

Problem 65 (MCQ — Relational Model Basics) *Which one of the following best describes the **degree** of a relation schema?*
A. Number of tuples currently stored
B. Number of attributes in the schema
C. Number of candidate keys
D. Number of foreign keys

Problem 66 (MCQ — Domain of Attributes) *Let attribute AGE have domain = integers between 0 and 120. Which constraint type enforces this rule?*
A. Referential integrity
B. Domain constraint
C. Key constraint
D. Entity integrity

Problem 67 (MSQ — Keys) *Consider relation R(A,B,C,D,E) with candidate keys $\{A,C\}$ and $\{B,D\}$. Which of the following are superkeys?*
A. A C
B. A B C
C. B D E
D. C D

Problem 68 (NAT — Superkeys Count) *Relation R has 5 attributes and exactly one candidate key consisting of 2 attributes. How many superkeys are possible?*

Problem 69 (MCQ — Integrity Constraints) *Which constraint ensures that every foreign key value must either be NULL or match a primary key value in the referenced relation?*
A. Domain constraint
B. Referential integrity
C. Entity integrity
D. Tuple integrity

Problem 70 (MCQ — 1:N Mapping Rule) *For a binary 1:N relationship without attributes, the correct mapping is:*
A. Create a new relation for relationship
B. Add foreign key of N-side into 1-side
C. Add foreign key of 1-side into N-side
D. Merge both entity tables always

Problem 71 (MCQ — 1:1 Mapping with Total Participation) *A binary 1:1 relationship has total participation from both entities and no attributes. Minimum tables required:*
A. 1
B. 2
C. 3
D. Depends on keys

Problem 72 (MCQ — Weak Entity Mapping) *A weak entity W with owner E and partial key P is mapped to:*
A. Table with only P
B. Table with PK(E)+P
C. Separate table without FK
D. Merge into owner always

Problem 73 (MSQ — Relationship Table Contents) *Which are included in a relationship table for an M:N relationship with attributes?*
A. Primary keys of participating entities
B. Non-key attributes of entities
C. Relationship attributes
D. Foreign keys referencing entities

Problem 74 (NAT — Tables Count) *Two strong entities connected by one M:N relationship with two attributes. No other constraints. Minimum number of tables after mapping:*

Problem 75 (MCQ — Recursive Relationship) *A recursive 1:N relationship is mapped by:*
A. Always new table
B. Self foreign key on N-side
C. Self foreign key on 1-side
D. Two new tables

Problem 76 (MCQ — n-ary Relationship) *An n-ary relationship among strong entities is mapped into:*
A. Merge into one entity table
B. Multiple binary tables only
C. One separate relation with all PKs as composite key
D. No table required

Problem 77 (MSQ) *Which of the following are domain constraint violations?*
A. Storing a string in an integer attribute
B. Inserting duplicate primary key
C. Value outside permitted range
D. Foreign key referencing missing tuple

Problem 78 (MCQ) *In a relation, entity integrity constraint implies:*
A. Primary key cannot be NULL
B. Foreign key cannot be NULL
C. All attributes must be unique
D. All domains must be numeric

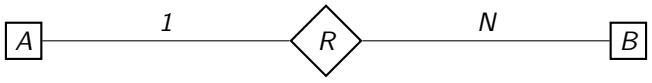
Problem 79 (MCQ) *A foreign key in a relation can reference:*
A. Only primary key of another relation
B. Any attribute of another relation
C. A candidate key of another relation
D. Only composite keys

Problem 80 (MSQ) *Which of the following operations can violate referential integrity (no cascade rules assumed)?*
A. Insert into child with unmatched FK
B. Delete referenced parent tuple
C. Update referenced parent key value
D. Insert into parent table

Problem 81 (MCQ) *A domain in the relational model refers to:*
A. Set of tuples in a relation
B. Set of allowed values for an attribute
C. Set of keys
D. Set of relations

Problem 82 (NAT) *Relation R has 5 attributes. Attribute A is the only key attribute. How many non-empty superkeys exist?*

Problem 83 (MCQ) *Consider the ER diagram below.*



R has no attributes. Minimum number of tables after mapping is:
A. 1
B. 2
C. 3
D. 4

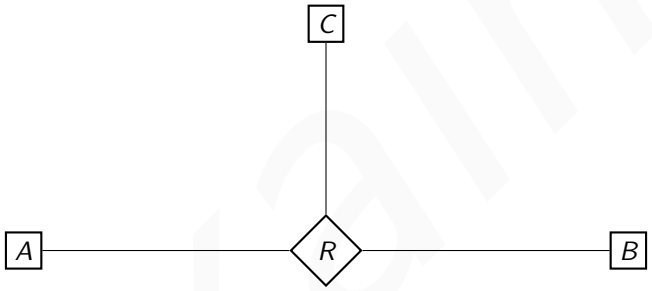
Problem 84 (MCQ) *For a binary M:N relationship with no relationship attributes, the mapping requires:*
A. One table
B. Two tables
C. Three tables
D. Four tables

Problem 85 (MCQ) *A weak entity set is mapped to:*
A. No table
B. A table with only its attributes
C. A table including owner key as part of primary key
D. A relationship table only

Problem 86 (NAT) *A weak entity has a partial key with 2 attributes and depends on a strong entity with a single-attribute key. Primary key size after mapping = _____*

Problem 87 (MCQ) *An n-ary relationship (n>2) in ER mapping results in:*
A. No separate table
B. Merge into one entity
C. Separate relation with FKs to all entities
D. Two relations only

Problem 88 (MCQ) *Consider the ternary ER diagram.*



Minimum number of tables required:
A. 2
B. 3
C. 4
D. 5

Problem 89 (MCQ) *If a binary relationship is 1:1 and both sides have total participation, it can be mapped using:*
A. Three tables
B. Two tables only
C. One merged table
D. Four tables

Problem 90 (MCQ) *If a binary relationship is 1:N with total participation on N-side and no attributes, mapping requires:*
A. Separate relationship table
B. FK on N-side entity table
C. FK on 1-side entity table
D. Two relationship tables

Problem 91 (MSQ) *A binary relationship has attributes and is M:N. Which are true?*
A. Needs separate table
B. Table includes both entity keys
C. Relationship attributes go into that table
D. Can always be merged into one entity table

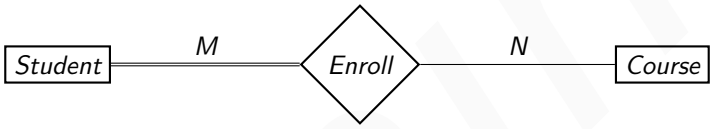
Problem 92 (MCQ) *Total participation of an entity in a relationship implies:*
A. FK must be UNIQUE
B. FK must be NOT NULL
C. FK must be composite
D. FK must be indexed

Problem 93 (MCQ) *Partial participation implies:*
A. FK must be NULL always
B. FK may be NULL
C. FK must be primary key
D. No FK allowed

Problem 94 (MCQ) *Consider recursive relationship (Employee manages Employee) with 1:N. Mapping requires:*
A. New table only
B. Self-referencing foreign key
C. Two separate entity tables
D. No foreign key

Problem 95 (NAT) *Two entities are connected by two different M:N relationships without attributes. Minimum number of tables after mapping = ____*

Problem 96 (MCQ) *Consider ER diagram.*



Enroll has no attributes. Student has total participation. Tables required:
A. 2
B. 3
C. 4
D. 5

Problem 97 (MCQ) *If both entities in a binary relationship have partial participation and ratio is 1:1 with no attributes, mapping needs:*
A. One table
B. Two tables with FK in either one
C. Three tables
D. No FK needed

Problem 98 (MSQ) *In ER to relational mapping, relationship table primary key for ternary relationship typically includes:*
A. Key of entity 1
B. Key of entity 2
C. Key of entity 3
D. Only one chosen key

Problem 99 (MCQ) *Relationship with its own attributes in 1:N mapping is usually implemented by:*
A. Dropping attributes
B. Adding them to N-side table
C. Adding them to 1-side table
D. Creating weak entity

Problem 100 (MCQ) *Which mapping never creates a separate table if no relationship attributes exist?*
A. M:N
B. 1:N
C. n-ary
D. Weak identifying

Problem 101 (MCQ) *In a relational database, the "Degree" of a relation refers to the number of _____, while the "Cardinality" refers to the number of _____.*
A. Attributes, Tuples
B. Tuples, Attributes
C. Domains, Columns
D. Keys, Relationships

Problem 102 (MSQ) *Which of the following are properties of a **Relation** in the relational model?*
A. The order of tuples is not significant.
B. The order of attributes is not significant.
C. All attribute values must be atomic.
D. Every relation must have at least one candidate key.

Problem 103 (MCQ) *Consider the following ER diagram:*

*Both entities have simple primary keys. $E2$ has **total participation** in relationship R. What is the minimum number of tables required to represent this in the relational model?*
A. 1
B. 2
C. 3
D. 4

Problem 104 (NAT) *Consider a relationship R between E_1 (Many) and E_2 (Many). Each entity E_1 and E_2 has a simple primary key. E_1 has 10 attributes, E_2 has 8 attributes, and R has 3 attributes. All attributes are simple and single-valued. The number of attributes in the table specifically representing relationship R in the relational model is _____.*

Problem 105 (MCQ) *In the context of referential integrity, if Table B has a foreign key referencing Table A, which of the following operations on Table A can violate the constraint?*
A. Insertion of a new tuple.
B. Deletion of an existing tuple.
C. Updating a non-key attribute.
D. None of the above.

Problem 106 (NAT) *A relation R has 4 attributes $\{A, B, C, D\}$. If $\{A\}$ and $\{B\}$ are the only candidate keys, the total number of superkeys is _____.*

Problem 107 (NAT) *Let $R(A, B, C, D)$ be a relation where A is the primary key and C is a foreign key referring to A with `ON DELETE CASCADE`. The relation has tuples: $\{(1, x, 1, y), (2, y, 1, z), (3, z, 2, x)\}$. If tuple $(1, x, 1, y)$ is deleted, the number of tuples remaining in the relation is _____.*

Problem 108 (MCQ) *A "Primary Key" is a _____ that has been chosen by the database designer as the principal means of identifying tuples.*
A. Superkey
B. Candidate Key
C. Foreign Key
D. Partial Key

Problem 109 (NAT) *An entity E has a **multi-valued attribute** M. E has 20 tuples. If 5 tuples have 2 values for M, 10 tuples have 3 values for M, and 5 tuples have 1 value for M, how many tuples will the table for M contain? _____.*

Problem 110 (MSQ) *Which of the following operations can violate **Referential Integrity**?*
A. Inserting into the referencing (child) table.
B. Deleting from the referencing (child) table.
C. Updating the primary key in the referenced (parent) table.
D. Deleting from the referenced (parent) table.

Problem 111 (NAT) *Given a relation $R(A, B, C, D)$, how many superkeys are possible if the only candidate keys are $\{AB\}$ and $\{CD\}$? _____.*

Problem 112 (MCQ) *In an ER diagram, a **double line** connecting an entity set to a relationship set indicates:*
A. Partial Participation
B. Total Participation
C. One-to-Many Cardinality
D. Derived Attribute

Problem 113 (NAT) *Consider a binary relationship R (1:N) between E_1 (1) and E_2 (N). E_1 has 3 attributes, E_2 has 5 attributes, and R has 2 attributes. If we merge the relationship into the table of E_2, how many attributes will the E_2 table have? _____.*

Problem 114 (MCQ) *Which of the following is **NOT** allowed in a relational database?*
A. Two rows having the same value for a non-key attribute.
B. A primary key attribute having a NULL value.
C. A foreign key attribute having a NULL value.
D. A table having more than one foreign key.

Problem 115 (NAT) *A database has two tables $T1(A, B)$ and $T2(C, D)$, where D is a foreign key referencing A. If $T1$ has 10 tuples and $T2$ has 15 tuples, the **maximum** number of distinct values in column D is _____.*

Problem 116 (MCQ) *What is the minimum number of tables needed to represent a self-referencing 1:N relationship (e.g., Employee manages Employees)?*
A. 1
B. 2
C. 3
D. 0

Problem 117 (NAT) *An n-ary relationship R connects n entity sets. If each entity set has a simple primary key, the table representing R in the relational model will have at least _____ attributes.*

Problem 118 (MCQ) *Which constraint ensures that "The age of an employee must be between 18 and 65"?*
A. Entity Integrity
B. Referential Integrity
C. Domain Constraint / Check Constraint
D. Key Constraint

Problem 119 (MSQ) *In a relation instance, which of the following are impossible?*
A. Two identical tuples.
B. A tuple with more attributes than the schema.
C. A primary key value that appears in two different tuples.
D. A foreign key value that does not exist in the referenced table.

Problem 120 (NAT) *Consider a relation with 3 attributes $\{A, B, C\}$. If there are NO candidate keys (i.e., every attribute is non-unique and can be NULL), how many superkeys are there? _____.*

3.10 GATE PYQs

GATEPYQ 8 *Which one of the options given below refers to the degree (or arity) of a relation in relational database systems?* ***GATE CSE 2023***

A. Number of attributes of its relation schema.
B. Number of tuples stored in the relation.
C. Number of entries in the relation.
D. Number of distinct domains of its relation schema.

GATEPYQ 9 *Consider the following statements S1 and S2 about the relational data model:*
S1: A relation scheme can have at most one foreign key.
S2: A foreign key in a relation scheme R cannot be used to refer to tuples of R.
Which one of the following choices is correct? ***GATE CSE 2021***

A. Both S1 and S2 are true
B. S1 is true and S2 is false
C. S1 is false and S2 is true
D. Both S1 and S2 are false

GATEPYQ 10 *Which of the following is NOT a superkey in a relational schema with attributes V, W, X, Y, Z and primary key VY?* ***GATE CSE 2016***

A. VXYZ
B. VWXZ
C. VWXY
D. VWXYZ

GATEPYQ 11 *An ER model of a database consists of entity types A and B. These are connected by a relationship R which does not have its own attribute. Under which one of the following conditions, can the relational table for R be merged with that of A?* ***GATE CSE 2017***

A. Relationship R is one-to-many and the participation of A in R is total
B. Relationship R is one-to-many and the participation of A in R is partial
C. Relationship R is many-to-one and the participation of A in R is total
D. Relationship R is many-to-one and the participation of A in R is partial

GATEPYQ 12 *Given an instance of the STUDENTS relation as shown below*

StudentID	StudentName	StudentEmail	StudentAge	CPI
2345	Shankar	shankar@math	X	9.4
1287	Swati	swati@ee	19	9.5
7853	Shankar	shankar@cse	19	9.4
9876	Swati	swati@mech	18	9.3
8765	Ganesh	ganesh@civil	19	8.7

Student Details

For (StudentName, StudentAge) to be a key for this instance, the value x should NOT be equal to_. ***GATE CSE 2014***

GATEPYQ 13 *The maximum number of super keys for the relation schema R (E, F, G, H) with E as the key is _____.* ***GATE CSE 2014***

GATEPYQ 14 *Consider a relational table with a single record for each registered student with the following attributes.*

1. *Registration_Number: Unique registration number for each registered student*
2. *UID: Unique Identity number, unique at the national level for each citizen*

3. BankAccount_Number: Unique account number at the bank. A student can have multiple accounts or joint accounts. This attribute stores the primary account number

4. Name: Name of the Student

5. Hostel_Room: Room number of the hostel

Which of the following options is INCORRECT? ***GATE CSE 2011***

A. BankAccount_Number is a candidate key

B. Registration_Number can be a primary key

C. UID is a candidate key if all students are from the same country

D. If S is a superkey such that $S \cap \{UID\}$ is NULL then $S \cup \{UID\}$ is also a superkey

GATEPYQ 15 *Let E1 and E2 be two entities in an E/R diagram with simple single-valued attributes. R1 and R2 are two relationships between E1 and E2, where R1 is one-to-many and R2 is many-to-many. R1 and R2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model?* ***GATE CSE 2005***

A. 2

B. 3

C. 4

D. 5

GATEPYQ 16 *Consider the following tables T1 and T2.*

T1

P	Q
2	2
3	8
7	3
5	8
6	9
8	5
9	8

T2

R	S
2	2
8	3
3	2
9	7
5	7
7	2

In table T1, P is the primary key and Q is the foreign key referencing R in table T2 with ondelete cascade and on-update cascade. In table T2, R is the primary key and S is the foreign key referencing P in table T1 on-delete set NULL and on-update cascade. In order to delete record (3,8) from table T1, the number of additional records that need to be deleted from table T1 is _____. ***GATE CSE 2017***

GATEPYQ 17 *The following table has two attributes A and C where A is the primary key and C is the foreign key referencing A with on-delete cascade. The set of all tuples that must be additionally deleted to preserve referential integrity when the tuple (2,4) is deleted is:* ***GATE CSE 2005***

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

A. (3,4) and (6,4)

B. (5,2) and (7,2)

C. (5,2), (7,2) and (9,5)

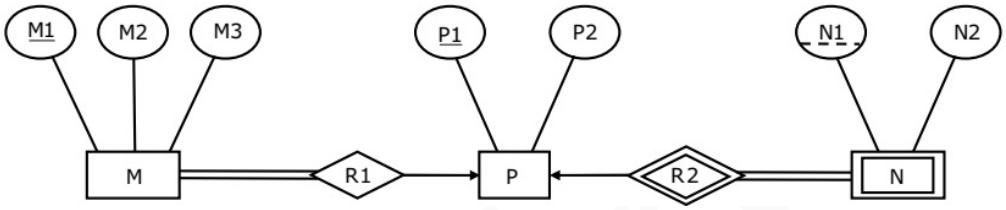
D. (3,4), (4,3) and (6,4)

GATEPYQ 18 *Let R(a, b, c) and S(d, e, f) be two relations in which d is the foreign key of S that refers to the primary key of R. Consider the following four operations on R and S:*

I. Insert into R
II. Insert into S
III. Delete from R
IV. Delete from S
Which of the following can cause violation of the referential integrity constraint above? ***GATE CSE 1997***

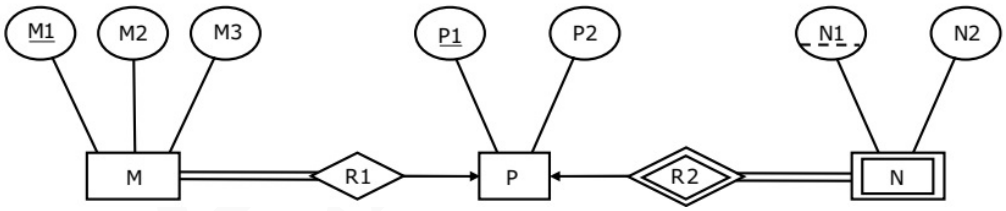
A. Both I and IV
B. Both II and III
C. All of these
D. None of these

GATEPYQ 19 *Consider the following ER diagram. Which of the following is a correct attribute set for one of the tables for the minimum number of tables needed to represent M, N, P, R1, R2?* ***GATE CSE 2008***



A. {M1,M2,M3,P1}
B. {M1,P1,N1,N2}
C. {M1,P1,N1}
D. {M1,P1}

GATEPYQ 20 *Consider the following ER diagram. The minimum number of tables needed to represent M, N, P, R1, R2 is* ***GATE CSE 2008***



A. 2
B. 3
C. 4
D. 5

3.11 Try it Yourself

Exercise 80 (MCQ — Relation Instance vs Schema) *Consider a relation schema STUDENT(RollNo, Name, Dept, Age). Over time, new student records are inserted and some are deleted. Which of the following components changes as these operations occur, while the schema itself remains unchanged?*
A. Number of attributes of STUDENT
B. Domains of attributes
C. Set of tuples in the relation
D. Attribute names

Exercise 81 (MCQ — Composite Key Meaning) *In the relational model, a primary key is said to be composite when:*
A. It contains at least one foreign key attribute
B. It is formed using more than one attribute together
C. It references multiple relations
D. It is derived from another key

Exercise 82 (MSQ — Candidate Key Properties) *Which of the following statements about candidate keys in a relation are correct?*
A. Every candidate key uniquely identifies tuples
B. A candidate key must be minimal
C. A relation can have more than one candidate key
D. A candidate key may contain NULL values

Exercise 83 (NAT — Counting Superkeys) *Consider a relation R(A,B,C,D) where the only candidate key is the attribute set $\{A,B\}$. How many superkeys does relation R have?* ____

Exercise 84 (MCQ — Foreign Key Reference Rule) *In a relational database, a foreign key defined in a relation R1 must reference which of the following in relation R2?*
A. Any indexed attribute of R2
B. Only the primary key of R2
C. A candidate key of R2
D. Any non-key attribute of R2

Exercise 85 (MCQ — Mapping Multivalued Attribute) *An entity BOOK has a multivalued attribute Author. While converting the ER model to relational schema, how should this attribute be mapped?*
A. Store all authors as comma-separated values in BOOK table
B. Create a separate table containing Book key and Author
C. Convert Author into a weak entity only
D. Ignore the multivalued attribute

Exercise 86 (NAT — Multivalued Attribute Rows) *An entity has 15 tuples. A multivalued attribute associated with it has exactly 2 values for each tuple. After ER to relational mapping, how many rows will appear in the table created for this multivalued attribute?* ____

Exercise 87 (MCQ — Derived Attribute Concept) *In an ER model, an attribute Age is derived from DateOfBirth. Which statement best describes such a derived attribute?*
A. It must always be stored physically in the database
B. It is computed from other stored attributes
C. It must be part of the primary key
D. It represents a multivalued attribute

Exercise 88 (MSQ — Domain Constraint Enforcement) *Which of the following mechanisms help enforce domain constraints on an attribute?*
A. Data type specification
B. CHECK constraint
C. Value range restriction
D. Foreign key declaration

Exercise 89 (MCQ — 1:N Relationship with Attributes) *Consider a binary 1:N relationship between entities A and B that has its own attributes. During ER-to-relational mapping, where are the relationship attributes stored?*
A. In a new separate relationship table always
B. In the table corresponding to the N-side entity
C. In the table corresponding to the 1-side entity
D. They are not stored

Exercise 90 (MCQ — Participation and NULL Foreign Key) *If an entity set has partial participation in a relationship, what does this imply about the foreign key introduced during mapping?*
A. It must be UNIQUE
B. It must be NOT NULL
C. It may be NULL
D. It must be composite

Exercise 91 (NAT — Attribute Count After Mapping) *Entity A has 5 attributes and entity B has 7 attributes. They are connected by a 1:N relationship having 2 attributes, and the relationship is mapped into the N-side table. How many attributes will the N-side table contain after mapping?* ____

Exercise 92 (MCQ — Weak Entity Partial Key) *In a weak entity set, the partial key has which uniqueness property?*
A. It is globally unique across all entities
B. It is unique only within each owner entity
C. It is always a single attribute
D. It is always a foreign key

Exercise 93 (MSQ — Weak Entity Requirements) *Which of the following are necessary characteristics of a weak entity set?*
A. It depends on an owner entity set
B. It has total participation in identifying relationship
C. It has a partial key
D. It has an independent primary key

Exercise 94 (NAT — Weak Entity Key Size) *A strong entity has a primary key of 2 attributes. Its weak entity has a partial key of 2 attributes. What is the total number of attributes in the primary key of the weak entity table after mapping?* ____

Exercise 95 (MCQ — Ternary Relationship Mapping) *A ternary relationship among three strong entities has two descriptive attributes. How is it mapped into the relational model?*
A. Merge into one of the entity tables
B. Create three binary relationship tables
C. Create one separate table with three foreign keys and the relationship attributes
D. No separate table is required

Exercise 96 (MSQ — Referential Integrity Violations) *Assume no cascade options are defined. Which of the following operations can violate referential integrity?*
A. Inserting a tuple in child table with unmatched foreign key
B. Deleting a referenced tuple from parent table
C. Updating a referenced primary key value in parent table
D. Inserting a new tuple into parent table

Exercise 97 (NAT — Cascade Delete Effect) *A parent table row is referenced by 4 rows in a child table. If ON DELETE CASCADE is specified and the parent row is deleted, how many child rows will also be deleted?* ____

Exercise 98 (MCQ — Cardinality Meaning) *In the relational model, the cardinality of a relation refers to:*
A. Number of attributes
B. Number of tuples currently stored
C. Number of candidate keys
D. Number of domains

Exercise 99 (MCQ — Superkey vs Candidate Key) *Which of the following correctly distinguishes a candidate key from a superkey?*
A. A candidate key is always composite, but a superkey is not
B. A candidate key is a minimal superkey
C. A superkey cannot contain extra attributes
D. A candidate key may not be unique

Exercise 100 (NAT — Superkeys Multiple Keys) *Consider a relation R(A, B, C) that has two candidate keys: A and B. Based on this information, how many superkeys does the relation R have?*
Total superkeys = ____

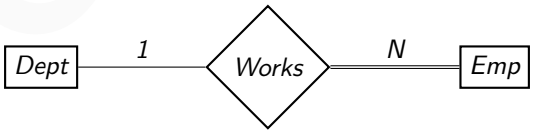
Exercise 101 (MCQ — ER Participation Symbol) *In an Entity–Relationship diagram, what does a double line drawn between an entity set and a relationship set represent?*
A. Weak entity
B. Total participation
C. Identifying relationship
D. Multivalued attribute

Exercise 102 (MCQ — Relationship Without Attributes) *When a binary 1:N relationship has no attributes of its own, how is it mapped into relational tables?*
A. Three tables are created
B. Two tables are created with a foreign key on the N-side
C. A new relationship table is created
D. A weak entity table is created

Exercise 103 (NAT — Tables from ER) *An ER design contains 4 strong entity sets and 2 many-to-many relationships, and none of the relationships have their own attributes. After mapping this ER model to relational schema, how many tables will be created?*
Tables after mapping = ____

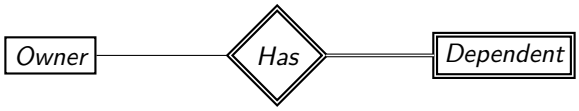
Exercise 104 (MSQ — Relation Properties) *Which of the following properties are satisfied by relations in the relational model?*
A. No duplicate tuples are allowed
B. Attribute values are atomic
C. Rows are stored in a specific order
D. Attributes are named

Exercise 105 (MCQ — ER Mapping Count) *Consider the following ER diagram showing a relationship between Department and Employee.*



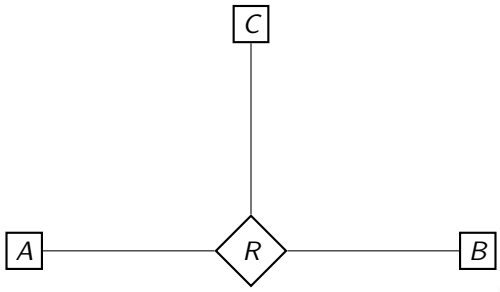
Assume the relationship has no attributes. How many tables are required after mapping this ER diagram to relations?
A. 1
B. 2
C. 3
D. 4

Exercise 106 (MCQ — Weak Entity Diagram) *Consider the following ER diagram that shows a weak entity Dependent related to an Owner through an identifying relationship.*



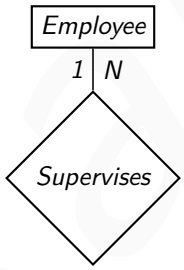
What is the minimum number of tables required after mapping this ER diagram to relational schema?
A. 1
B. 2
C. 3
D. 4

Exercise 107 (NAT — Ternary Diagram Tables) *Consider the following ER diagram that shows a ternary relationship R among three entity sets A, B, and C.*



What is the minimum number of tables required after mapping this ternary relationship into relations?
Minimum tables = ____

Exercise 108 (MCQ — Recursive Diagram) *Consider the following ER diagram showing a recursive relationship Supervises on the Employee entity set.*



Assume the relationship has no attributes. How many tables are needed after mapping?
A. 1
B. 2
C. 3
D. 4

Exercise 109 (MSQ — Composite Attribute Mapping) *Suppose an entity has a composite attribute Address that consists of components City and Zip. How is such a composite attribute handled during ER-to-relational mapping?*
A. Only the component attributes are stored
B. The composite attribute is stored as a single field only
C. No separate table is required for the composite attribute
D. The component attributes become separate columns

Exercise 110 (NAT — Composite Expansion) *An entity has 5 simple attributes and 1 composite attribute that consists of 3 component attributes. After mapping to a relational table, how many columns will be stored in total?*
Columns stored = ____

Exercise 111 (MCQ — M:N With Attributes) *How is a binary many-to-many (M:N) relationship that has its own attributes mapped into the relational model?*
A. By placing a foreign key on one side only
B. By creating a separate table with foreign keys and relationship attributes
C. By merging the two entity tables
D. By converting it into a weak entity

Exercise 112 (NAT — M:N Table Attributes) *Entity set A has a primary key of size 1 attribute, and entity set B has a primary key of size 2 attributes. Their many-to-many relationship has 3 attributes of its own. How many columns will the relationship table contain after mapping?*
Columns in relationship table = ____

Exercise 113 (MCQ — Key Minimality) *A superkey is classified as a candidate key under which condition?*
A. It is indexed
B. It is minimal
C. It is numeric
D. It is a foreign key

Exercise 114 (MSQ — Superkey Facts) *Which of the following statements are true about superkeys in a relation?*
A. They may include extra attributes
B. They must be minimal
C. They always ensure uniqueness
D. They are supersets of candidate keys

Exercise 115 (NAT — Superkeys Single Key) *A relation has 6 attributes, and the only candidate key is a single attribute A. How many superkeys does this relation have?*
Superkeys = ____

Exercise 116 (MCQ — FK Null Case) *Under which participation constraint is a foreign key attribute allowed to take NULL values?*
A. Total participation
B. Partial participation
C. Identifying participation
D. Weak participation

Exercise 117 (MCQ — Identifying Relationship) *An identifying relationship in an ER model is specifically associated with which type of entity set?*
A. Strong entities
B. Weak entities
C. Multivalued attributes
D. Derived attributes

Exercise 118 (NAT — Tables with Weak + M:N) *An ER design contains two strong entity sets, one weak entity set dependent on the first strong entity, and one many-to-many relationship between the two strong entities. How many tables will be produced after mapping?*
Tables = ____

Exercise 119 (MSQ — Atomic Domain) *In the relational model, what does it mean for an attribute domain to be atomic?*
A. Values are indivisible
B. There are no repeating groups
C. Multivalued attributes are allowed
D. Values are stored as scalars

Exercise 120 (MCQ — Relationship Degree) *What is the degree of a relationship that involves four different entity sets?*
A. Binary
B. Ternary
C. Quaternary
D. Recursive

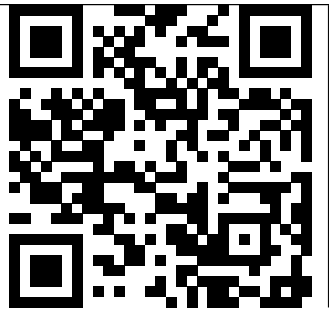
Exercise 121 (NAT — n-ary FK Count) *When an n-ary relationship is mapped into a relation, what is the minimum number of foreign keys that must be included in the resulting table?*

____

Exercise 122 (MCQ — Duplicate Tuples) *Why does the relational model not allow duplicate tuples in a relation?*
A. Because of domain constraints
B. Because of key constraints
C. Because relations follow set semantics
D. Because of referential integrity constraints

3.12 YouTube Links and QR Codes

Lecture	Details	YouTube Link	QR Code
11	Introduction to Relational Model — Degree — Cardinality — Domain — Atomicity — NULL Values	https://youtu.be/2_AuljBCtPw	
12	KEYS: Composite — Candidate — Primary — Unique — Super — Foreign — Alternate Keys	https://youtu.be/GnOa7f43liI	
13	Integrity Constraints — Domain Constraint — Key Constraint — Entity Constraint — Referential Constraint	https://youtu.be/MBQFiM6CqJ8	
14	Mapping ER Model to Relational Model — Rules 1–4 Explained	https://youtu.be/8b-flBBl3S8	

15	Mapping ER Model to Relational Model — Rules 5–7 Explained	https://youtu.be/sJbcz1XgSEA	
16	Mapping ER Model to Relational Model — Rules 8–10 Explained	https://youtu.be/rBCdBoheAQE	
17	Examples on Mapping ER Model to Relational Model	https://youtu.be/jQoGml59ai0	
18	Solutions to Practice Problems on Relational Model	https://youtu.be/v6tFEFkdmzw	
19	Solved GATEPYQs on Relational Model	https://youtu.be/xSbc4yenX-Y	

Chapter 4

Relational Algebra & Tuple Calculus

Query Language

A **query language** is a language used by a user to request information from a database. These languages generally operate at a **higher level of abstraction** than conventional programming languages. Query languages are broadly classified into **imperative**, **functional**, and **declarative** types.

Imperative query language

In an **imperative query language**, the user specifies an explicit **sequence of operations** that the system must execute on the database to produce the required result. Such languages typically use **state variables** whose values change during the execution process.
Examples: Relational Algebra, Procedural SQL (SQL with cursors and loops).

Functional query language

In a **functional query language**, computation is written as the **evaluation of functions** applied to database data or to the outputs of other functions. These functions are **side-effect free** and do not modify the program state.
Examples: Datalog, Lisp-based query systems.

Declarative query language

In a **declarative query language**, the user states **what information is needed** rather than describing the steps or procedures to retrieve it. The requirement is usually expressed using **mathematical logic**, and the database system itself determines the method to obtain the result.
Examples: SQL, Relational Calculus (Tuple/Domain Relational Calculus).

4.1 Relational Algebra

What is Relational Algebra?

Relational Algebra is a **procedural query language** used in DBMS to operate on relations (tables). It defines a set of **formal operations** to retrieve and manipulate data. It is important because it forms the **theoretical foundation of SQL** and query optimization in relational databases.

Relational Operations & Symbols

Operation	Symbol	One-line Explanation
Select Operation	σ	Chooses rows that satisfy a given condition (row filtering).
Project Operation	π	Extracts specific columns from a relation (column filtering).
Cartesian-Product Operation	$\times$	Combines every tuple of one relation with every tuple of another relation.
Join Operation	$\bowtie$	Combines related tuples from two relations based on a join condition.
Union	$\cup$	Returns tuples that appear in either of the two relations (duplicates removed).
Set Difference	$-$	Returns tuples that appear in the first relation but not in the second.
Intersection	$\cap$	Returns tuples common to both relations.
Assignment Operation	$\leftarrow$	Stores the result of a relational expression into a temporary relation.
Rename Operation	ρ	Renames a relation or its attributes for clarity or reuse.

4.1.1 Selection Operation (σ)

The Select Operation (σ) — Definition and Purpose

The **Select operation** in Relational Algebra is used to **filter rows (tuples)** from a relation that satisfy a given condition called a **predicate**. It does not change the attributes (columns); it only reduces the number of tuples (rows).
It is denoted by the lowercase Greek letter **sigma** (σ).
General form:

$$\sigma_{\text{predicate}}(\text{Relation})$$

The predicate is written as a subscript to σ, and the relation name is written in parentheses.

Example 75: Instructor Relation — Base Table

Assume the relation **instructor**:

ID	Name	Dept_name	Salary
101	Amit	Physics	95000
102	Neha	CSE	88000
103	Ravi	Physics	72000
104	Sara	Math	99000

To select tuples where an attribute equals a specific value, we use the equality operator.

$$\sigma_{\text{dept\_name} = \text{“Physics”}}(\text{instructor})$$

This returns only those instructors who belong to the Physics department. Result:

ID	Name	Dept_name	Salary
101	Amit	Physics	95000
103	Ravi	Physics	72000

Selection Using Comparison Operators

Selection predicates may use comparison operators:

$$=,\ \neq,\ <,\ \leq,\ >,\ \geq$$

Example form:

$$\sigma_{salary>90000}(\text{instructor})$$

This filters tuples based on numeric comparison.

Example 76: Select instructors with salary greater than 90000

Result:

ID	Name	Dept_name	Salary
101	Amit	Physics	95000
104	Sara	Math	99000

Combining Conditions with Logical Connectives

Multiple predicates can be combined using logical connectives:

$$\wedge\ (\text{AND}),\quad \vee\ (\text{OR}),\quad \neg\ (\text{NOT})$$

Example:

$$\sigma_{\text{dept\_name} = \text{“Physics”} \wedge salary>90000}(\text{instructor})$$

Only tuples satisfying **all** conditions are selected when AND is used.

Example 77: Physics instructors with salary > 90000

Result:

ID	Name	Dept_name	Salary
101	Amit	Physics	95000

Comparison Between Two Attributes

The selection predicate can also compare **two attributes** of the same relation.
Form:

$$\sigma_{\text{Attribute1} = \text{Attribute2}}(\text{Relation})$$

This selects tuples where the two attribute values are equal.

Example 78: Department table — attribute comparison

Assume relation **department**:

Dept_name	Building
Physics	Physics
CSE	TechBlock
Math	Math

Query:

$$\sigma_{\text{dept\_name} = \text{building}}(\text{department})$$

Result keeps only rows where both attributes match.

4.1.2 Projection Operation (Π)

The Project Operation (Π) — Definition and Purpose

The **Project operation** is used to **select specific attributes (columns)** from a relation and discard the remaining attributes. It is a **unary operation** that works on a single relation.
Projection does not filter rows based on conditions — instead, it reduces the number of columns in the result.
Since a relation is a **set of tuples**, any **duplicate rows are automatically removed** after projection.
Projection is denoted by the uppercase Greek letter **Pi** (Π).
General form:

$$\Pi_{A_1, A_2, \ldots, A_n}(\text{Relation})$$

Only the listed attributes appear in the output relation.

Example 79: Instructor Relation — Base Table

Assume the relation **instructor**:

ID	Name	Dept_name	Salary
101	Srinivasan	CSE	65000
102	Wu	Physics	90000
103	Mozart	Music	40000
104	Einstein	Physics	95000
105	Gold	CSE	65000

Suppose we want only **ID, Name, and Salary** and do not care about department. We use projection to keep only these attributes.

$$\Pi_{\text{ID, Name, Salary}}(\text{instructor})$$

The Dept_name column is removed in the result. Result:

ID	Name	Salary
101	Srinivasan	65000
102	Wu	90000
103	Mozart	40000
104	Einstein	95000
105	Gold	65000

Duplicate Elimination in Projection

After projection, if two or more tuples become identical on the selected attributes, only **one copy** is kept because relations do not allow duplicate tuples.
Projection therefore performs **duplicate elimination automatically**.

Example 80: Projection causing duplicates to be removed

$$\Pi_{\text{Salary}}(\text{instructor})$$

Result (note that salary 65000 appears once even if multiple instructors have it):

Salary
65000
90000
40000
95000

Generalized Projection with Expressions

The basic projection operator allows only attribute names. A **generalized projection** allows **expressions on attributes**.
This is useful when computed values are required in the result.
Example:

$$\Pi_{\text{ID, Name, Salary/12}}(\text{instructor})$$

This computes derived attributes during projection.

Example 81: Projection with computed monthly salary

$$\Pi_{\text{ID, Name, Salary/12}}(\text{instructor})$$

Result (monthly salary):

ID	Name	Salary/12
101	Srinivasan	5416.67
102	Wu	7500
103	Mozart	3333.33
104	Einstein	7916.67
105	Gold	5416.67

4.1.3 Composition of Relational Operations

Composition of Relational Operations — Meaning and Need

A key property of Relational Algebra is that the **result of every operation is itself a relation**. Because of this closure property, the output of one operation can be used directly as the input to another operation.
This ability to combine multiple relational operations into a single expression is called **composition of relational operations**.
It allows us to build **complex queries step by step** using simpler operations.

Composed Expression Form

In a composed relational-algebra expression, instead of giving a base relation name as input to an operator, we can give a **relational expression** that evaluates to a relation.
General pattern:

$$\text{OuterOp}(\text{InnerOp}(\text{Relation}))$$

This is similar to arithmetic composition such as:

$$(3+5)\times 2$$

where the result of one operation becomes the input to another.

Example 82: Instructor Relation

ID	Name	Dept_name	Salary
101	Srinivasan	CSE	65000
102	Wu	Physics	90000
103	Mozart	Music	40000
104	Einstein	Physics	95000
105	Gold	CSE	65000

Query: **Find the names of all instructors in the Physics department.**
This requires two steps:

- First select Physics department tuples
- Then project only the Name attribute

Relational algebra expression:

$$\Pi_{\text{Name}}\left(\sigma_{\text{dept\_name} = \text{“Physics”}}(\text{instructor})\right)$$

Inner operation: selection
Outer operation: projection

Step 1 — Selection:

$$\sigma_{\text{dept\_name} = \text{“Physics”}}(\text{instructor})$$

ID	Name	Dept_name	Salary
102	Wu	Physics	90000
104	Einstein	Physics	95000

Step 2 — Projection on result:

$$\Pi_{\text{Name}}(\cdot)$$

Final Result:

Name
Wu
Einstein

Why Composition Matters

Composition is powerful because it enables:

- Multi-step query construction
- Reuse of intermediate results
- Formal query optimization by DBMS

- Direct mapping to nested SQL queries

All complex relational queries are built by composing basic operations like σ, Π, $\bowtie$, $\times$, and set operations.

4.1.4 The Cartesian-Product Operation ($\times$)

The Cartesian-Product Operation ($\times$)

The **Cartesian-product operation** combines tuples from two relations in **all possible pairs**. It is denoted by the cross symbol $\times$.
If r_1 and r_2 are two relations, their Cartesian product is written as:

$$r_1 \times r_2$$

Each tuple of r_1 is concatenated with every tuple of r_2 to form a new tuple. Thus, the result relation contains attributes from **both** relations.

Size of Cartesian Product

If relation r_1 has n_1 tuples and relation r_2 has n_2 tuples, then:

$$|r_1 \times r_2| = n_1 \times n_2$$

So the result can become very large. Because of this, Cartesian product is usually followed by a **selection** condition (which leads to a join).

Tuple Construction Rule

Unlike the mathematical Cartesian product that produces ordered pairs (t_1, t_2), relational algebra **concatenates** the two tuples into a single longer tuple containing attributes of both relations.
Schema of result is the **concatenation of schemas**.

Example 83: Base Relations — instructor and teaches

instructor

ID	Name	Dept
101	Srinivasan	CSE
102	Wu	Physics
103	Einstein	Physics

teaches

ID	Course
101	CS101
103	PH201

Result of instructor $\times$ teaches

instr.ID	Name	Dept	teaches.ID	Course
101	Srinivasan	CSE	101	CS101
101	Srinivasan	CSE	103	PH201
102	Wu	Physics	101	CS101
102	Wu	Physics	103	PH201
103	Einstein	Physics	101	CS101
103	Einstein	Physics	103	PH201

Number of tuples $= 3 \times 2 = 6$

Attribute Name Conflicts and Prefixing

If both relations contain attributes with the same name (like ID), we must **prefix** them with relation names to avoid ambiguity.
Example schema:

$$(\text{instructor.ID}, \text{Name}, \text{Dept}, \text{teaches.ID}, \text{Course})$$

If an attribute appears in only one relation, the prefix may be dropped when no confusion arises.

Requirement of Distinct Relation Names

The two operand relations must have **distinct names**. Problems arise when:

- A relation is multiplied with itself
- A derived expression has no relation name

This is handled using the **rename operation** (ρ), which assigns a new temporary name to a relation before applying Cartesian product.

Practical Use

Cartesian product alone is rarely used in queries because it produces many unrelated tuple combinations. In practice it is almost always followed by a **selection condition**, forming a **join** operation.

4.1.5 Rename Operator (ρ)

The Rename Operation (ρ)

The **Rename operation** assigns a new name to:

- a relation (result of an expression), or
- the attributes of a relation

Relational-algebra expressions normally produce unnamed intermediate results. The rename operator ρ allows us to reference them later.
Symbol: lowercase Greek rho — ρ
Two forms exist:

(1) Rename Relation Only

$$\rho_X(E)$$

Result of expression E is given relation name X.

(2) Rename Relation + Attributes

$$\rho_{X(A_1,A_2,\ldots,A_n)}(E)$$

Result is named X and its attributes are renamed.

Why Rename is Needed

Rename is required when:

- Same relation is used multiple times (self join)
- Intermediate results must be referenced
- Attribute names clash
- Derived attributes need meaningful names

Most common use: **self joins and comparison queries.**

Example 84: Instructor Relation — Comparison Query

ID	Name	Dept	Salary
10101	Srinivasan	CSE	65000
12121	Wu	Physics	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000

Query: Find ID and Name of instructors who earn more than the instructor with ID = 12121.
Strategy:

- Create two copies of the relation via renaming.
- Let $I1$ represent all instructors.
- Let $I2$ represent the specific instructor with ID 12121.
- Perform a Cartesian product and filter where $I1.Salary > I2.Salary$.

Relational Algebra Expression:

$$I2 \leftarrow \sigma_{ID=12121}(\text{Instructor})$$

$$\Pi_{I1.ID,I1.Name}(\sigma_{I1.Salary>I2.Salary}(\rho_{I1}(\text{Instructor}) \times I2))$$

Final Query Result:

ID	Name
22222	Einstein

4.1.6 Assignment Operator (←)

Assignment Operation (←)

The **Assignment operation** allows us to store the result of a relational-algebra expression into a **temporary relation variable**.
Symbol used: ←
It works like variable assignment in programming languages.

Syntax:

$$X \leftarrow E$$

where:

- E = relational algebra expression
- X = temporary relation name

The assigned relation can be reused in later expressions.

Why Assignment is Used

Assignment improves query writing by:

- Breaking complex queries into steps
- Improving readability
- Avoiding repeated expressions
- Enabling modular query construction
- Supporting intermediate result reuse

Assignment itself does **not display output**. Only the final expression result is shown.

Example 85: Assignment Operation — Stepwise Query Using Cartesian Product

Employee(EmpID, Name, DeptID)

EmpID	Name	DeptID
1	Asha	10
2	Ravi	20
3	Meera	10

Department(DeptID, DeptName)

DeptID	DeptName
10	CSE
20	Physics
30	Math

[Step 1 — Assignment with Cartesian Product]

$$T \leftarrow Employee \times Department$$

Temporary relation T stores the Cartesian product.

EmpID	Name	E.DeptID	D.DeptID	DeptName
1	Asha	10	10	CSE
1	Asha	10	20	Physics
1	Asha	10	30	Math
2	Ravi	20	10	CSE
2	Ravi	20	20	Physics
2	Ravi	20	30	Math
3	Meera	10	10	CSE
3	Meera	10	20	Physics
3	Meera	10	30	Math

[Step 2 — Assignment with Selection]

$$M \leftarrow \sigma_{E.DeptID=D.DeptID}(T)$$

Keep only matching department rows.

EmpID	Name	E.DeptID	D.DeptID	DeptName
1	Asha	10	10	CSE
2	Ravi	20	20	Physics
3	Meera	10	10	CSE

[Step 3 — Assignment with Projection]

$$Result \leftarrow \Pi_{Name,DeptName}(M)$$

Final projected result.

Name	DeptName
Asha	CSE
Ravi	Physics
Meera	CSE

4.1.7 Join Operation (⋈)

The Join Operation (⋈)

The **Join operation** is used to combine related tuples from two relations based on a **matching condition**. It brings together information stored in different tables using a common attribute.
Join is needed because a plain Cartesian product pairs every tuple with every other tuple, which creates many incorrect combinations.
Join keeps only those tuple pairs that satisfy a specified condition.

Join as Selection on Cartesian Product

A join can be formally defined using Cartesian product followed by selection.
For relations $r(R)$ and $s(S)$ with condition θ:

$$r \bowtie_\theta s \;=\; \sigma_\theta(r \times s)$$

So join is a **derived operation**:

- First compute Cartesian product
- Then apply selection condition

Example 86: Base Relations — instructor and teaches

instructor

ID	Name	Dept
101	Srinivasan	CSE
102	Wu	Physics
103	Einstein	Physics
104	Gold	Math

teaches

ID	Course_id
101	CS101
101	CS102
103	PH201

Query: **Find instructor information together with course ids of courses they taught.**
Cartesian product alone is incorrect because it pairs every instructor with every course.
Correct pairs occur only where:

$$\text{instructor.ID} = \text{teaches.ID}$$

Using selection on product:

$$\sigma_{\text{instructor.ID} \,=\, \text{teaches.ID}}(\text{instructor} \times \text{teaches})$$

Using join operator:

$$\text{instructor} \bowtie_{\text{instructor.ID} = \text{teaches.ID}} \text{teaches}$$

Both expressions are equivalent.

instr.ID	Name	Dept	Course_id
101	Srinivasan	CSE	CS101
101	Srinivasan	CSE	CS102
103	Einstein	Physics	PH201

Instructor 102 and 104 do not appear because they have no matching tuple in **teaches**.

Duplicate Attribute Columns After Join

After join, common attributes (like ID from both relations) may appear twice. We usually remove redundancy using projection.
Example:

$$\Pi_{\text{Name, Dept, Course\_id}}(\text{instructor} \bowtie_{\text{instructor.ID} = \text{teaches.ID}} \text{teaches})$$

Join Types

- Theta Join — any comparison condition
- Equi Join — equality condition only
- Natural Join — equi join + duplicate column removal
- Left Outer — keep all left tuples
- Right Outer — keep all right tuples
- Full Outer — keep all tuples from both sides
- Self Join — relation joined with itself

Example 87: Base Relations Used in All Join Examples

Employee(EmpID, Name, DeptID)

EmpID	Name	DeptID
1	Asha	10
2	Ravi	20
3	Meera	30
4	John	NULL

Department(DeptID, DeptName)

DeptID	DeptName
10	CSE
20	Physics
40	Math

1. Theta Join ($\bowtie_\theta$)

Theta join uses **any comparison operator** $(=, \neq, <, >, \leq, \geq)$ in the join condition.
General form:

$$R \bowtie_\theta S$$

Definition:

$$R \bowtie_\theta S = \sigma_\theta(R \times S)$$

Example 88: Theta Join with equality condition

$$\text{Employee} \bowtie_{\text{Employee.DeptID} = \text{Department.DeptID}} \text{Department}$$

EmpID	Name	E.DeptID	D.DeptID	DeptName
1	Asha	10	10	CSE
2	Ravi	20	20	Physics

Non-matching tuples are removed.

2. Equi Join

Equi join is a **special case of theta join** where the condition uses **only equality (=)** comparisons.
Duplicate join columns are **not automatically removed**.
Symbol is still $\bowtie_{\text{condition}}$.

Example 89: Equi Join Output Table

Equi join condition:

$$\text{Employee} \bowtie_{\text{Employee.DeptID} = \text{Department.DeptID}} \text{Department}$$

Output relation (both join attributes are retained):

EmpID	Name	Employee.DeptID	Department.DeptID	DeptName
1	Asha	10	10	CSE
2	Ravi	20	20	Physics

Notice that:

- Meera (DeptID 30) is excluded — no matching department
- John (NULL DeptID) is excluded — no match

- DeptID 40 (Math) is excluded — no matching employee
- Both DeptID columns appear — projection can remove one if needed

3. Natural Join (⋈)

Natural join:

- automatically joins on attributes with the **same name**
- uses equality comparison
- **removes duplicate join columns** in the output

No explicit condition is written.

$$Employee \bowtie Department$$

Example 90: Natural Join Result

EmpID	Name	DeptID	DeptName
1	Asha	10	CSE
2	Ravi	20	Physics

Only one DeptID column appears.

4. Left Outer Join (⟕)

Left outer join keeps:

- all matching tuples
- **all tuples from left relation**
- fills NULLs where right side has no match

Example 91: Left Outer Join

$$Employee \leftouterjoin Department$$

EmpID	Name	DeptID	DeptName
1	Asha	10	CSE
2	Ravi	20	Physics
3	Meera	30	NULL
4	John	NULL	NULL

5. Right Outer Join ($\rtimes\!\bowtie$)

Right outer join keeps:

- all matching tuples
- all tuples from right relation
- NULLs where left side has no match

Example 92: Right Outer Join — Full Output Table

Right outer join expression:

$$Employee \rtimes\!\bowtie Department$$

All departments are preserved even if no employee belongs to them.

EmpID	Name	DeptID	DeptName
1	Asha	10	CSE
2	Ravi	20	Physics
NULL	NULL	40	Math

Explanation:

- Dept 10 $\rightarrow$ matched with Asha
- Dept 20 $\rightarrow$ matched with Ravi
- Dept 40 $\rightarrow$ no employee $\rightarrow$ Employee attributes become NULL
- Meera (DeptID 30) and John (NULL DeptID) are excluded — not preserved in right outer join

6. Full Outer Join ($\!\rtimes\!\bowtie\!\ltimes\!$)

Full outer join keeps:

- all matching tuples
- all non-matching tuples from both relations
- fills NULLs on whichever side is missing

Example 93: Full Outer Join — Full Output Table

Full outer join expression:

$$Employee \!\rtimes\!\bowtie\!\ltimes\! Department$$

EmpID	Name	DeptID	DeptName
1	Asha	10	CSE
2	Ravi	20	Physics
3	Meera	30	NULL
4	John	NULL	NULL
NULL	NULL	40	Math

Explanation:

- Asha and Ravi — matching rows
- Meera — no department match → DeptName = NULL
- John — DeptID is NULL → no match → DeptName = NULL
- DeptID 40 (Math) — no employee → Employee columns are NULL

7. Self Join

A self join joins a relation with itself. Rename operator ρ is used to create aliases.
Used for hierarchical or comparison queries.

Example 94: Self Join — Employees in Same Department (Output)

$$\rho_{E1}(Employee) \bowtie_{E1.DeptID=E2.DeptID} \rho_{E2}(Employee)$$

E1.EmpID	E1.Name	E2.EmpID	E2.Name	DeptID
1	Asha	1	Asha	10
2	Ravi	2	Ravi	20
3	Meera	3	Meera	30

Notes:

- Each employee matches with themselves (same DeptID)
- No cross pairs appear because no two employees share a DeptID
- John is excluded because NULL does not match with NULL in join condition

4.1.8 SET Operations ($\cap \cup -$)

Union Operation ($\cup$)

The **union** operation combines tuples from two relations and removes duplicates.
Rules for union:

1. Both relations must have the same number of attributes (same arity).

2. Corresponding attributes must have the same type.
3. Result contains all tuples that appear in either relation.

Example 95: Union Example — Students in Two Clubs

Data:
ChessClub(StudentID, Name)

StudentID	Name
1	Alice
2	Bob
3	Carol

DramaClub(StudentID, Name)

StudentID	Name
2	Bob
4	Dave
5	Eve

Query: Find all students who are in either Chess or Drama Club:

$$AllStudents \leftarrow ChessClub \cup DramaClub$$

Result:

StudentID	Name
1	Alice
2	Bob
3	Carol
4	Dave
5	Eve

Intersection Operation ($\cap$)

The **intersection** operation finds tuples common to both relations.
Rules for intersection:

1. Both relations must have the same number of attributes (same arity).
2. Corresponding attributes must have the same type.
3. Result contains only tuples present in both relations.

Example 96: Intersection Example — Students in Both Clubs

Using the same data as above:

$$CommonStudents \leftarrow ChessClub \cap DramaClub$$

Result:

StudentID	Name
2	Bob

Set Difference Operation ($-$)

The **set difference** operation finds tuples in one relation but not in the other.
Rules for difference:

1. Both relations must have the same number of attributes (same arity).
2. Corresponding attributes must have the same type.
3. Result contains tuples in the first relation that are not in the second.

Example 97: Difference Example — Students only in Chess Club

Using the same data as above:

$$ChessOnly \leftarrow ChessClub - DramaClub$$

Result:

StudentID	Name
1	Alice
3	Carol

Handling Duplicates in Set Operations

In relational algebra, relations are **sets of tuples**, not multisets. This means:

- **Duplicates are automatically removed** in all set operations: Union ($\cup$), Intersection ($\cap$), and Difference ($-$).
- Even if a tuple appears multiple times in a relation, it is considered only once.
- The result of a set operation will contain each distinct tuple at most once.

Example 98: Duplicates in Union

Relations:
A(StudentID, Name)

StudentID	Name
1	Alice
2	Bob
2	Bob
3	Carol

B(StudentID, Name)

StudentID	Name
2	Bob
3	Carol
4	Dave

Union: $A \cup B$

StudentID	Name
1	Alice
2	Bob
3	Carol
4	Dave

Intersection: $A \cap B$

StudentID	Name
2	Bob
3	Carol

Difference: $A - B$

StudentID	Name
1	Alice

Note: Even though Bob appears twice in relation A, duplicates are removed automatically in all operations.

4.1.9 Division Operator

Division Operator (÷) — Meaning and Use

The division operator is used to answer queries of the form:
*"Find all X such that X is related to **all** Y."*
Formally, given two relations:

- $R(A, B)$ — the dividend
- $S(B)$ — the divisor

The expression:

$$R \div S$$

returns a relation with attribute(s) A containing all values of A that are associated with **every value of B in S.**
Rules / Requirements:

- The attributes of S must be a subset of the attributes of R.
- The resulting relation will have attributes $R - S$.
- Duplicates are removed automatically (relations are sets).

Example 99: Division Operator — Simple Example

Relations:
Enrolled(Student, Course)

Student	Course
Alice	Math
Alice	Physics
Bob	Math
Bob	Physics
Bob	Chemistry
Carol	Math
Carol	Physics

Required Courses:
Required(Course)

Course
Math
Physics

Query: Find students who are enrolled in **all required courses**.

Enrolled ÷ Required

Result:

Student
Alice
Bob
Carol

Explanation:

- Alice is enrolled in Math & Physics - satisfies all required courses.
- Bob is enrolled in Math & Physics & Chemistry - satisfies all required courses.
- Carol is enrolled in Math & Physics - satisfies all required courses.
- Only courses in the **Required** relation are considered.

4.1.10 Operators: Min/Max Tuples in Relational Algebra

Operator	Min Tuples	Max Tuples	Explanation
Cartesian ($\times$)	$\lvert R\rvert \cdot \lvert S\rvert$	$\lvert R\rvert \cdot \lvert S\rvert$	Every tuple of R pairs with every tuple of S, fixed number.
Theta / Equi Join ($\bowtie_\theta$)	0	$\lvert R\rvert \cdot \lvert S\rvert$	Min zero if no match; max all pairs satisfy join condition.
Natural Join ($\bowtie$)	0	$\sum_{v \in V}$(count of v in R $\cdot$ count of v in S)	Min zero if no common values; max depends on multiplicities of join attributes.
Left Outer Join ($=\!\!\bowtie$)	$\lvert R\rvert$	$\lvert R\rvert \cdot \lvert S\rvert$	All R tuples appear at least once; max if all R $\times$ S match.
Right Outer Join ($\bowtie\!\!=$)	$\lvert S\rvert$	$\lvert R\rvert \cdot \lvert S\rvert$	All S tuples appear at least once; max if all R $\times$ S match.
Full Outer Join ($=\!\!\bowtie\!\!=$)	$\max(\lvert R\rvert, \lvert S\rvert)$	$\lvert R\rvert \cdot \lvert S\rvert$	All tuples from both sides appear at least once; max if all R $\times$ S match.

4.2 Examples on Relation Algebra

Example 100: Suppliers-Parts-Catalog Relational Algebra Queries

Schema:

- **Suppliers**(sid: integer, sname: string, address: string)
- **Parts**(pid: integer, pname: string, color: string)
- **Catalog**(sid: integer, pid: integer, cost: real)

Queries and Solutions:

1. **Find the names of suppliers who supply some red part.**

$$\Pi_{sname}\Big((\sigma_{color='Red'}(Parts)) \bowtie Catalog \bowtie Suppliers\Big)$$

Result: Anil, Balu, Deepak

2. **Find the sids of suppliers who supply some red or green part.**

$$\Pi_{sid}\Big((\sigma_{color='Red' \vee color='Green'}(Parts)) \bowtie Catalog\Big)$$

Result: 1, 2, 3, 4

3. **Find the sids of suppliers who supply some red part or are at Bengaluru.**

$$\Pi_{sid}(\sigma_{color='Red'}(Parts) \bowtie Catalog) \cup \Pi_{sid}(\sigma_{address='Bengaluru'}(Suppliers))$$

Result: 1, 2, 4

4. **Find the sids of suppliers who supply some red part and some green part.**

$$(\Pi_{sid}(\sigma_{color='Red'}(Parts) \bowtie Catalog)) \cap (\Pi_{sid}(\sigma_{color='Green'}(Parts) \bowtie Catalog))$$

Result: 1

5. **Find the sids of suppliers who supply every part.**

$$\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(Parts)$$

Result: None

6. **Find the sids of suppliers who supply every red part.**

$$\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(\sigma_{color='Red'}(Parts))$$

Result: 2 (Supplies 101 and 103)

7. **Find the sids of suppliers who supply every red or green part.**

$$\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(\sigma_{color='Red' \vee color='Green'}(Parts))$$

Result: None (No supplier provides 101, 102, and 103)

8. **Find the sids of suppliers who supply every red part or supply every green part.**

$$(\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(\sigma_{color='Red'}(Parts))) \cup (\Pi_{sid,pid}(Catalog) \div \Pi_{pid}(\sigma_{color='Green'}(Parts)))$$

Result: 1, 2, 3

9. **Find pairs of sids (s_1, s_2) such that s_1 charges more for some part than s_2 for that same part.**

$$\Pi_{C1.sid,C2.sid}(\sigma_{C1.pid=C2.pid \wedge C1.cost>C2.cost}(\rho_{C1}(Catalog) \times \rho_{C2}(Catalog)))$$

10. **Find the pids of parts supplied by every supplier at less than \$200.**

$$\Pi_{pid}(Catalog) - \Pi_{pid}(\sigma_{cost \geq 200}(Catalog))$$

Result: 101, 102, 103, 104

Example 101: Author Book and Publication Queries

Schema:

- **author**(<u>author_id</u>, first_name, last_name)
- **author_pub**(<u>author_id, pub_id</u>, author_position)
  - author_id → author(author_id)
  - pub_id → pub(pub_id)
- **book**(<u>book_id</u>, book_title, month, year, editor)
  - editor → author(author_id)
- **pub**(<u>pub_id</u>, title, book_id)
  - book_id → book(book_id)

Questions and Solutions:

1. **Query:** $\Pi_{book\_title}(book)$
   Solution: Projection returns all distinct book titles.

   $$\text{tuples returned} = |\text{distinct book\_title values in book}|$$

2. **Query:** $|\Pi_{author\_id}(author) - \Pi_{editor}(book)|$
   Solution: Set difference finds authors who are not editors. Cardinality counts them.

   Answer: Number of authors who have never acted as a book editor.

3. **Query:** Names of all authors who are book editors
   Solution:

   $$\Pi_{first\_name,last\_name}(author \bowtie_{author.author\_id=book.editor} book)$$

   This joins authors with books where they are listed as editors, and returns their names.

4. **Query:** Find author_ids of authors who have a publication in **every** book.
   Solution (Division): First, we need a relation matching authors to books via their publications, then divide by all book IDs.

   $$\text{Let } R = \Pi_{author\_id,book\_id}(author\_pub \bowtie pub)$$

   $$\text{Let } S = \Pi_{book\_id}(book)$$

   $$\text{Final Query: } R \div S$$

   Note: The dividend R must contain both author_id and book_id so the result is ({author_id, book_id} - {book_id}) = {author_id}.

5. **Query:** Find names of authors who have published in every book edited by 'John Doe'.
   Solution:

   $$\text{TargetBooks} = \Pi_{book\_id}(\sigma_{first\_name='John' \wedge last\_name='Doe'}(author \bowtie_{author\_id=editor} book))$$

   $$\text{AuthorBookMap} = \Pi_{author\_id,book\_id}(author\_pub \bowtie pub)$$

   $$\text{Final Result: } \Pi_{first\_name,last\_name}(author \bowtie (AuthorBookMap \div TargetBooks))$$

4.3 Tuple Relational Calculus

Tuple Relational Calculus

Tuple Relational Calculus (TRC) is a **non-procedural (declarative) query language** where queries are written as logical formulas that describe **what result is required** rather than how to compute it. It is important because it provides the **formal logical basis** for declarative query languages like SQL.

Concept

- TRC is a **nonprocedural** query language: it specifies *what* information is desired rather than *how* to compute it (unlike relational algebra).
- A TRC query is expressed as:
$$\{t \mid P(t)\}$$
where t is a tuple variable and $P(t)$ is a predicate (condition) that must hold true.
- Notation:
  - $t[A]$ denotes the value of attribute A in tuple t.
  - $t \in r$ denotes that tuple t belongs to relation r.
- Logical constructs in TRC:
  - $\exists t \in r(Q(t))$: There exists a tuple t in relation r such that predicate $Q(t)$ is true.
  - $\forall t \in r(Q(t))$: For all tuples t in relation r, predicate $Q(t)$ is true.
  - $\neg P$: Logical NOT (negation)
  - $P \wedge Q$: Logical AND
  - $P \vee Q$: Logical OR
  - $P \Rightarrow Q$: Implication (equivalent to $\neg P \vee Q$)
- **Resulting relation attributes:** A tuple variable t only contains the attributes explicitly mentioned in the query. TRC respects set semantics, so duplicates are automatically removed.
- TRC expressions can involve multiple relations, connected with logical operators.

Operators and Symbols

Component / Operation	Symbol	One-line Explanation
Query Form	$\{t \mid P(t)\}$	Result is the set of tuples t for which predicate $P(t)$ is true.
Tuple Variable	t	Represents a tuple variable ranging over a relation.
Relation Membership	$t \in R$	Specifies that tuple variable t belongs to relation R.
Existential Quantifier	$\exists$	States that there exists at least one tuple satisfying a condition.
Universal Quantifier	$\forall$	States that all tuples must satisfy a condition.
Logical AND	$\wedge$	Both conditions must be true.
Logical OR	$\vee$	At least one condition must be true.
Logical NOT	$\neg$	Negates a condition.
Comparison Operators	$=, \neq, <, \leq, >, \geq$	Used to compare attribute values in predicates.
Attribute Reference	$t.A$	Refers to attribute A of tuple variable t.

Example 102:

Example 1: Find instructors with salary greater than $80,000
Relation schema:

- **instructor**(ID, name, dept_name, salary)

Sample data:

ID	Name	Dept_Name	Salary
101	Einstein	Physics	95000
102	Crick	Biology	85000
103	Gold	Chemistry	75000

TRC query:

$$\{t \mid \exists s \in instructor(t[ID] = s[ID] \wedge s[salary] > 80000)\}$$

Explanation:

- t is a tuple variable defined only on ID.
- s ranges over all tuples of instructor.
- Only instructors with salary > 80000 satisfy the predicate.

Result:

$$\{(101), (102)\}$$

Example 103:

Example 2: Find names of instructors whose department is in Watson building
Relation schemas:

- **instructor**(ID, name, dept_name, salary)
- **department**(dept_name, building)

Sample data:

instructor:

ID	Name	Dept_Name	Salary
101	Einstein	Physics	95000
102	Crick	Biology	85000
103	Gold	Chemistry	75000

department:

Dept_Name	Building
Physics	Watson
Biology	Feynman
Chemistry	Watson

TRC query:

$$\Big\{ t \;\Big|\; \exists s \in \text{instructor}\ \big(t[\text{name}] = s[\text{name}] \wedge \exists u \in \text{department}\ \big(u[\text{dept\_name}] = s[\text{dept\_name}]\ \wedge\ u[\text{building}] =' Watson')\big)\Big\}$$

Explanation:

- t is defined only on name.
- s ranges over instructors; u ranges over departments.
- Select instructors whose dept_name matches a department in Watson building.

Result:

$$\{\text{Einstein}, \text{Gold}\}$$

Example 104:

Example 3: Courses taught in Fall 2017 or Spring 2018
Relation schema:

- **section**(course_id, semester, year)

Sample data:

Course_ID	Semester	Year
CS-101	Fall	2017
CS-315	Fall	2017
CS-319	Spring	2018
CS-347	Spring	2018
CS-101	Spring	2018

TRC query (union / OR):

$$\Big\{t \;\Big|\; \big(\exists s \in \text{section}\ (t[\text{course\_id}] = s[\text{course\_id}] \ \wedge\ s[\text{semester}] =' Fall' \ \wedge\ s[\text{year}] = 2017)\big) \\ \vee \big(\exists u \in \text{section}\ (t[\text{course\_id}] = u[\text{course\_id}] \ \wedge\ u[\text{semester}] =' Spring' \ \wedge\ u[\text{year}] = 2018)\big)\Big\}$$

Result:

$$\{\text{CS-101}, \text{CS-315}, \text{CS-319}, \text{CS-347}\}$$

TRC query (intersection / AND): Courses offered in both Fall 2017 and Spring 2018:

$$\Big\{t \;\Big|\; \exists s \in \text{section}\ (t[\text{course\_id}] = s[\text{course\_id}] \ \wedge\ s[\text{semester}] =' Fall' \ \wedge\ s[\text{year}] = 2017) \\ \wedge\ \exists u \in \text{section}\ (t[\text{course\_id}] = u[\text{course\_id}] \ \wedge\ u[\text{semester}] =' Spring' \ \wedge\ u[\text{year}] = 2018)\Big\}$$

Result:

$$\{\text{CS-101}\}$$

Example 105:

Example 4: Students who have taken all courses in Biology
Relation schemas:

- **student**(ID, name)
- **course**(course_id, dept_name)
- **takes**(ID, course_id)

TRC query:

$$\Big\{t \;\Big|\; \exists r \in \text{student}\ (r[\text{ID}] = t[\text{ID}]) \\ \wedge\ \forall u \in \text{course}\ \big(u[\text{dept\_name}] =' Biology' \ \Rightarrow\ \exists s \in \text{takes}\ (t[\text{ID}] = s[\text{ID}] \ \wedge\ s[\text{course\_id}] = u[\text{course\_id}])\big)\Big\}$$

Explanation:

- t ranges over students (ID only).
- For every Biology course u, there must exist a tuple s in takes showing that t has taken it.
- Subtlety: If no Biology courses exist, all student IDs satisfy the condition (hence first $\exists r$ clause is needed).

Safety of Tuple Relational Calculus Expressions

A tuple-relational-calculus expression may generate an infinite relation if unrestricted. For example:

$$\{t \mid \neg(t \in \text{instructor})\}$$

produces infinitely many tuples not in instructor, many of which contain values that do not appear in the database. Such expressions are **unsafe**.
To formally define safe expressions, we introduce the **domain of a tuple relational formula** P, denoted dom(P).

- dom(P) is the set of all values referenced by P, including:
  - Constants explicitly mentioned in P, and
  - Values appearing in tuples of any relation mentioned in P.
- An expression $\{t \mid P(t)\}$ is **safe** if all values that appear in the result are from dom(P).

Examples:

- dom$(t \in \text{instructor} \wedge t[\text{salary}] > 80000)$ is the set containing 80000 and all values appearing in the instructor relation.
- dom$(\neg(t \in \text{instructor}))$ is also the set of all values appearing in instructor, but the expression itself is unsafe because it could generate tuples containing values outside dom$(\neg(t \in \text{instructor}))$.

Key Point: Safe expressions are guaranteed to produce finite results, whereas unsafe expressions may produce infinite results. Therefore, the class of allowed tuple-relational-calculus expressions is restricted to safe expressions.

4.4 Problems

Problem 121 *Consider the relations:* ***Employee****(eid, ename, dept),* ***Salary****(eid, amount)*

Which of the following relational algebra expressions returns the names of employees earning more than $50,000? (MSQ)

A. $\Pi_{ename}(\sigma_{amount>50000}(Employee \bowtie Salary))$
B. $\Pi_{ename}(\sigma_{amount\geq 50000}(Employee \bowtie Salary))$
C. $\Pi_{ename}(\sigma_{amount<50000}(Employee \bowtie Salary))$
D. $\Pi_{ename}(Employee) - \Pi_{ename}(\sigma_{amount\leq 50000}(Employee \bowtie Salary))$

Problem 122 *Relations:* ***Student****(sid, sname),* ***Course****(cid, cname),* ***Enroll****(sid, cid)*

The following query finds students enrolled in all courses:

$$T \leftarrow \Pi_{cid}(Course), \quad Result \leftarrow Student \div T$$

If there are 5 students and 4 courses, and each student is enrolled in all courses, the number of tuples returned is: (NAT)

Problem 123 *Relations:* ***Supplier****(sid, sname, city),* ***Part****(pid, pname, color),* ***Catalog****(sid, pid, cost)*

Which relational algebra expressions return sids of suppliers who supply both red and blue parts? (MSQ)

A. $\Pi_{sid}(\sigma_{color='red'}(Catalog \bowtie Part)) \cap \Pi_{sid}(\sigma_{color='blue'}(Catalog \bowtie Part))$
B. $\Pi_{sid}(\sigma_{color='red' \wedge color='blue'}(Catalog \bowtie Part))$
C. $\Pi_{sid}(\sigma_{color='red'}(Catalog \bowtie Part)) \cup \Pi_{sid}(\sigma_{color='blue'}(Catalog \bowtie Part))$
D. $\Pi_{sid}(Catalog \div \Pi_{pid}(\sigma_{color='red' \vee color='blue'}(Part)))$

Problem 124 *Relations:* ***Employee****(eid, ename),* ***Dependent****(did, eid, dname, age)*

What does the relational algebra query

$$\Pi_{eid}(Employee \bowtie \sigma_{age>20}(Dependent))$$

return? (MCQ)

A. Employees with all dependents older than 20
B. Employees with at least one dependent older than 20
C. Employees with no dependent older than 20
D. All employees

Problem 125 *Relations:* ***Book****(bid, title, editor),* ***Author****(aid, name),* ***AuthorPub****(aid, pid)*

Which RA expression finds authors who are also editors? (MSQ)

A. $\Pi_{name}(Author \bowtie_{aid=editor} Book)$
B. $\Pi_{name}(Author) - \Pi_{name}(Book)$
C. $\Pi_{name}(Author \bowtie AuthorPub)$
D. $\Pi_{name}(\sigma_{aid=editor}(Author \bowtie Book))$

Problem 126 *Relations:* ***Student****(sid, sname),* ***Marks****(sid, subject, score)*

Consider the following relational algebra expression:

$$R1 \leftarrow \Pi_{sid,score}(\sigma_{subject='DBMS'}(Marks))$$
$$R2 \leftarrow \Pi_{S1.sid}(\rho_{S1}(R1) \bowtie_{S1.score<S2.score} \rho_{S2}(R1))$$
$$Result \leftarrow \Pi_{sid}(R1) - R2$$

If there are 10 students who appeared for the DBMS exam, and their scores are $\{88, 95, 72, 95, 84, 95, 90, 95, 60, 78\}$*, the number of tuples returned by the query is: (NAT)*

Problem 127 *Relations:* ***Employee****(eid, name, salary, did),* ***Department****(did, dname)*

Which of the following Relational Algebra expressions returns the IDs of departments where ***every*** *employee earns more than $50,000? (MSQ)*

A. $\Pi_{did}(Department) - \Pi_{did}(\sigma_{salary \leq 50000}(Employee))$
B. $\Pi_{did}(Employee) - \Pi_{did}(\sigma_{salary \leq 50000}(Employee))$
C. $\Pi_{did,eid}(Employee) \div \Pi_{eid}(\sigma_{salary > 50000}(Employee))$
D. $\Pi_{did}(\sigma_{salary > 50000}(Employee))$

Problem 128 *Relations:* ***R****(A,B),* ***S****(B,C), B primary key in S, foreign key in R*

Which of the following is NOT equivalent to $\sigma_{B>10}(R \bowtie S)$*? (MCQ)*

A. $\sigma_{B>10}(R) \bowtie S$
B. R *LOJ* $\sigma_{B>10}(S)$
C. $\sigma_{B>10}(R$ *LOJ* $S)$
D. $\sigma_{B>10}(S)$ *LOJ* R

Problem 129 *Consider the relations:* ***Employee****(eid, ename, dept),* ***Project****(pid, pname),* ***WorksOn****(eid, pid)*

Which relational algebra expression returns the eids of employees who work on every project in dept 'D1'? (MCQ)

A. $\Pi_{eid}(Employee \div \Pi_{pid}(\sigma_{dept='D1'}(Project \bowtie WorksOn)))$
B. $\Pi_{eid}(\sigma_{dept='D1'}(Employee) \div \Pi_{pid}(Project))$
C. $\Pi_{eid}(\sigma_{dept='D1'}(WorksOn))$
D. $\Pi_{eid}(Employee \bowtie WorksOn \bowtie \sigma_{dept='D1'}(Project))$

Problem 130 *Relations:* ***Student****(sid, sname),* ***Course****(cid, cname),* ***Enroll****(sid, cid)*

The following query:

$$T_1 \leftarrow \Pi_{cid}(\sigma_{sname='Alice'}(Enroll \bowtie Student)), \quad Result \leftarrow Enroll \div T_1$$

What does $Result$ *represent? (MCQ)*

A. Students who have taken all courses that Alice has taken
B. Courses that Alice has not taken
C. Students who have taken at least one course that Alice has taken
D. Students who have never taken the same courses as Alice

Problem 131 *Relations:* ***Product****(pid, pname, price)*

Consider the following relational algebra sequence:

$$R_1 \leftarrow \Pi_{pid,price}(Product)$$

$$R_2 \leftarrow \Pi_{S1.pid,S1.price}(\rho_{S1}(R_1) \bowtie_{S1.price<S2.price} \rho_{S2}(R_1))$$

$$R_3 \leftarrow R_1 - R_2$$

$$R_4 \leftarrow \Pi_{S3.pid,S3.price}(\rho_{S3}(R_2) \bowtie_{S3.price<S4.price} \rho_{S4}(R_2))$$

$$Result \leftarrow R_2 - R_4$$

If the prices of products in the table are $\{100, 200, 300, 300, 400, 400, 400\}$*, the number of tuples in the* $Result$ *is: (NAT)*

Problem 132 *Relations:* ***Project****(pid, ptype),* ***EmpProject****(eid, pid)*

Let $T_{AI} \leftarrow \Pi_{pid}(\sigma_{ptype='AI'}(Project))$*. The query is defined as:*

$$Result \leftarrow (EmpProject \div T_{AI}) - (EmpProject \div \Pi_{pid}(Project))$$

Suppose:

- *- Total projects = 10.*
- *- 'AI' type projects = 3.*
- *- Employee* E_1 *works on all 10 projects.*
- *- Employee* E_2 *works on only the 3 'AI' projects.*

- *- Employee E_3 works on 5 projects, including all 3 'AI' projects.*
- *- Employee E_4 works on only 2 'AI' projects.*

The number of tuples in $Result$ is: (NAT)

Problem 133 *Relations: **R**(A), **S**(B)*
Let R contain $\{1,2,3,4\}$ and S contain $\{1,2,3,4\}$. The expression is:

$$\Pi_A(\sigma_{A<B}(R \times S))$$

The number of tuples returned in the result set is: (NAT)

Problem 134 *Consider the relations:*

$$\textit{Student}(sid, sname, age), \quad \textit{Course}(cid, cname)$$

and an enrollment relation:

$$\textit{Enroll}(sid, cid)$$

Which tuple relational calculus query returns the names of students enrolled in all courses offered?
A. $\{s.sname \mid s \in \textit{Student} \wedge \forall c \in \textit{Course}(\exists e \in \textit{Enroll}(e.sid = s.sid \wedge e.cid = c.cid))\}$
B. $\{s.sname \mid s \in \textit{Student} \wedge \exists c \in \textit{Course}(\exists e \in \textit{Enroll}(e.sid = s.sid \wedge e.cid = c.cid))\}$
C. $\{s.sname \mid s \in \textit{Student} \wedge \exists e \in \textit{Enroll}(e.sid = s.sid)\}$
D. $\{s.sname \mid s \in \textit{Student} \wedge \forall e \in \textit{Enroll}(\exists c \in \textit{Course}(e.cid = c.cid))\}$

Problem 135 *Given relations:*

$$\textit{Employee}(eid, ename, deptid), \quad \textit{Department}(deptid, dname)$$

Which tuple relational calculus expression is safe for retrieving the names of employees in the 'HR' department?
A. $\{e.ename \mid e \in \textit{Employee} \wedge \exists d \in \textit{Department}(e.deptid = d.deptid \wedge d.dname =' HR')\}$
B. $\{e.ename \mid e \in \textit{Employee} \wedge \forall d \in \textit{Department}(e.deptid = d.deptid \wedge d.dname =' HR')\}$
C. $\{e.ename \mid e \in \textit{Employee} \wedge \neg\exists d \in \textit{Department}(e.deptid = d.deptid)\}$
D. $\{e.ename \mid e \in \textit{Employee} \wedge \forall d \in \textit{Department}(\neg(e.deptid = d.deptid))\}$

Problem 136 *Consider relations:*

$$\textit{Product}(pid, pname, price), \quad \textit{Order}(oid, pid)$$

Which TRC expression returns all product names that have not been ordered?
A. $\{p.pname \mid p \in \textit{Product} \wedge \neg\exists o \in \textit{Order}(o.pid = p.pid)\}$
B. $\{p.pname \mid p \in \textit{Product} \wedge \exists o \in \textit{Order}(o.pid = p.pid)\}$
C. $\{p.pname \mid p \in \textit{Product} \wedge \forall o \in \textit{Order}(o.pid \neq p.pid)\}$
D. Both A and C

Problem 137 *Given relations:*

$$\textit{Author}(aid, aname), \quad \textit{Book}(bid, title, aid)$$

Which tuple relational calculus query lists all authors who have written at least one book?
A. $\{a.aname \mid a \in \textit{Author} \wedge \exists b \in \textit{Book}(b.aid = a.aid)\}$
B. $\{a.aname \mid a \in \textit{Author} \wedge \forall b \in \textit{Book}(b.aid = a.aid)\}$
C. $\{a.aname \mid a \in \textit{Author} \wedge \neg\exists b \in \textit{Book}(b.aid = a.aid)\}$
D. $\{a.aname \mid a \in \textit{Author}\}$

4.5 GATE PYQs

GATEPYQ 21 *Consider the following two relations:*

$$R(A,B) = \begin{bmatrix} 10 & 20 \\ 20 & 30 \\ 30 & 40 \\ 30 & 50 \\ 50 & 95 \end{bmatrix}, \quad S(A,C) = \begin{bmatrix} 10 & 90 \\ 30 & 45 \\ 40 & 80 \end{bmatrix}$$

The total number of tuples obtained by evaluating the following expression:

$$\sigma_{B<C}(R \bowtie_{R.A=S.A} S)$$

is: ***GATE CSE 2024***

GATEPYQ 22 *The relation schema, Person(pid, city), describes the city of residence for every person uniquely identified by pid. The following relational algebra operators are available: selection, projection, cross product, and rename.*

To find the list of cities where at least 3 persons reside, using the above operators, the minimum number of cross product operations that must be used is ***GATE CSE 2024***

A. 1
B. 2
C. 3
D. 4

GATEPYQ 23 *Consider the following three relations in a relational database:*

Employee(eId, Name), Brand(bId, bName), Own(eId, bId)

Which of the following relational algebra expressions returns the set of eIds who own all the brands? ***GATE CSE 2022***

A. $\Pi_{eId}\big(\Pi_{eId,bId}(Own) \,/\, \Pi_{bId}(Brand)\big)$
B. $\Pi_{eId}(Own) - \Pi_{eId}\big((\Pi_{eId}(Own) \times \Pi_{bId}(Brand)) - \Pi_{eId,bId}(Own)\big)$
C. $\Pi_{eId}\big(\Pi_{eId,bId}(Own) \,/\, \Pi_{bId}(Own)\big)$
D. $\Pi_{eId}\big((\Pi_{eId}(Own) \times \Pi_{bId}(Own)) \,/\, \Pi_{bId}(Brand)\big)$

GATEPYQ 24 *A relation* $r(A,B)$ *in a relational database has 1200 tuples. The attribute* A *has integer values ranging from 6 to 20, and the attribute* B *has integer values ranging from 1 to 20. Assume that the attributes* A *and* B *are independently distributed.*

The estimated number of tuples in the output of

$$\sigma_{(A>10)\vee(B=18)}(r)$$

is: ***GATE CSE 2021***

GATEPYQ 25 *The following relation records the age of 500 employees of a company, where empNo (indicating the employee number) is the key:*

empAge(empNo, age)

Consider the following relational algebra expression:

$$\Pi_{empNo}\big(empAge \;\; \bowtie_{age>age1} \;\; \rho_{empNo1,age1}(empAge)\big)$$

What does the above expression generate? ***GATE CSE 2021***

A. Employee numbers of only those employees whose age is the maximum
B. Employee numbers of only those employees whose age is more than the age of exactly one other employee
C. Employee numbers of all employees whose age is not the minimum
D. Employee numbers of all employees whose age is the minimum

GATEPYQ 26 *Consider the following relations P(X, Y, Z), Q(X, Y, T), and R(Y, V):*

P		
X	Y	Z
X1	Y1	Z1
X1	Y1	Z2
X2	Y2	Z2
X2	Y4	Z4

Q		
X	Y	T
X2	Y1	2
X1	Y2	5
X1	Y1	6
X3	Y3	1

R	
Y	V
Y1	V1
Y3	V2
Y2	V3
Y2	V2

How many tuples will be returned by the following relational algebra query? ***GATE CSE 2019***

$$\Pi_X\left(\sigma_{P.Y=R.Y \wedge R.V=V2}(P \times R)\right) - \Pi_X\left(\sigma_{Q.Y=R.Y \wedge Q.T>2}(Q \times R)\right)$$

GATEPYQ 27 *Consider the relations $r(A, B)$ and $s(B, C)$, where $s.B$ is a primary key and $r.B$ is a foreign key referencing $s.B$. Consider the query*

$$Q : r \bowtie (\sigma_{B<5}(s))$$

Let LOJ denote the natural left outer-join operation. Assume that r and s contain no null values.
Which one of the following queries is NOT equivalent to Q? ***GATE CSE 2018***
A. $\sigma_{B<5}(r \bowtie s)$
B. $\sigma_{B<5}(r \text{ LOJ } s)$
C. $r \text{ LOJ } (\sigma_{B<5}(s))$
D. $\sigma_{B<5}(r) \text{ LOJ } s$

GATEPYQ 28 *Consider a database that has the relation schema $CR(StudentName, CourseName)$. An instance of the schema CR is as given below.*

StudentName	*CourseName*
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database:

$$T_1 \leftarrow \pi_{CourseName}\left(\sigma_{StudentName='SA'}(CR)\right)$$

$$T_2 \leftarrow CR \div T_1$$

The number of rows in T_2 is _____. ***GATE CSE 2017***

GATEPYQ 29 *Consider two relations $R_1(A, B)$ with the tuples $(1, 5), (3, 7)$ and $R_2(A, C)$ with the tuples $(1, 7), (4, 9)$. Assume that $R(A, B, C)$ is the full natural outer join of R_1 and R_2.*

Consider the following tuples of the form (A, B, C):

$$a = (1, 5, \textit{null}),\ b = (1, \textit{null}, 7),\ c = (3, \textit{null}, 9),\ d = (4, 7, \textit{null}),\ e = (1, 5, 7),\ f = (3, 7, \textit{null}),\ g = (4, \textit{null}, 9)$$

Which one of the following statements is correct? ***GATE CSE 2015***

A. R contains a, b, e, f, g but not c, d.
B. R contains all of a, b, c, d, e, f, g.
C. R contains e, f, g but not a, b.
D. R contains e but not f, g.

GATEPYQ 30 *Consider the relational schema given below, where eId of the relation dependent is a foreign key referring to $empId$ of the relation employee. Assume that every employee has at least one associated dependent in the dependent relation.*

employee(empId, empName, empAge)

dependent(depId, eId, depName, depAge)

Consider the following relational algebra query:

$$\Pi_{empId}\left(employee\right) - \Pi_{empId}\left(employee \bowtie_{(empId=eID)\wedge(empAge\leq depAge)} dependent\right)$$

The above query evaluates to the set of $empIds$ of employees whose age is greater than:
A. some dependent
B. all dependents
C. some of his/her dependents
D. all of his/her dependents ***GATE CSE 2014***

GATEPYQ 31 *What is the optimized version of the relational algebra expression*

$$\pi_{A_1}\left(\pi_{A_2}\left(\sigma_{F_1}\left(\sigma_{F_2}\left(r\right)\right)\right)\right)$$

where A_1, A_2 are sets of attributes in r with $A_1 \subset A_2$ and F_1, F_2 are Boolean expressions based on the attributes in r? ***GATE CSE 2014***
A. $\pi_{A_1}\left(\sigma_{F_1\wedge F_2}\left(r\right)\right)$
B. $\pi_{A_1}\left(\sigma_{F_1\vee F_2}\left(r\right)\right)$
C. $\pi_{A_2}\left(\sigma_{F_1\wedge F_2}\left(r\right)\right)$
D. $\pi_{A_2}\left(\sigma_{F_1\vee F_2}\left(r\right)\right)$

GATEPYQ 32 *Consider the following relations A, B and C:*

A

Id	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

B

Id	Name	Age
15	Shreya	24
25	Hari	40
98	Rohit	20
99	Rohit	11

C

Id	Phone	Area
10	2200	02
99	2100	01

How many tuples does the result of the following relational algebra expression contain? Assume that the schema of $A \cup B$ is the same as that of A. ***GATE CSE 2012***

$$(A \cup B) \bowtie \sigma_{A.Id>40 \vee C.Id<15}(C)$$

A. 7
B. 4
C. 5
D. 9

GATEPYQ 33 *Suppose $R_1(A, B)$ and $R_2(C, D)$ are two relation schemas. Let r_1 and r_2 be the corresponding relation instances. B is a foreign key that refers to C in R_2. If data in r_1 and r_2 satisfy referential integrity constraints, which of the following is ALWAYS TRUE?* ***GATE CSE 2012***

A. $\Pi_B(r_1) - \Pi_C(r_2) = \varnothing$
B. $\Pi_C(r_1) - \Pi_B(r_2) = \varnothing$
C. $\Pi_B(r_1) = \Pi_C(r_2)$
D. $\Pi_B(r_1) - \Pi_C(r_2) \neq \varnothing$

GATEPYQ 34 *The following functional dependencies hold for relations $R(A, B, C)$ and $S(B, D, E)$:*

$$B \rightarrow A, \quad A \rightarrow C$$

The relation R contains 200 tuples and the relation S contains 100 tuples. What is the maximum number of tuples possible in the natural join $R \bowtie S$? ***GATE CSE 2010***

A. 100
B. 200
C. 300
D. 2000

GATEPYQ 35 *Let R and S be two relations with the following schema:*

$$R(\underline{P, Q}, R_1, R_2, R_3), \quad S(\underline{P, Q}, S_1, S_2)$$

where $\{P, Q\}$ is the key for both schemas. Which of the following queries are equivalent? ***GATE CSE 2008***

I. $\Pi_P(R \bowtie S)$
II. $\Pi_P(R) \bowtie \Pi_P(S)$
III. $\Pi_P(\Pi_{P,Q}(R) \cap \Pi_{P,Q}(S))$
IV. $\Pi_P(\Pi_{P,Q}(R) - (\Pi_{P,Q}(R) - \Pi_{P,Q}(S)))$
A. Only I and II
B. Only I and III
C. Only I, II and III
D. Only I, III and IV

GATEPYQ 36 *Information about a collection of students is given by the relation*

$$\textit{studinfo}(\underline{studId}, name, sex)$$

The relation

$$\textit{enroll}(studId, courseId)$$

gives which student has enrolled for (or taken) what course(s). Assume that every course is taken by at least one male and at least one female student.

What does the following relational algebra expression represent? ***GATE CSE 2007***

$$\Pi_{\text{courseId}}\left(\left(\Pi_{\text{studId}}\left(\sigma_{\textit{sex}=\text{"female"}}\left(\text{studInfo}\right)\right) \times \Pi_{\text{courseId}}\left(\text{enroll}\right)\right) - \text{enroll}\right)$$

A. Courses in which all the female students are enrolled.
B. Courses in which a proper subset of female students are enrolled.
C. Courses in which only male students are enrolled.
D. None of the above

GATEPYQ 37 *Consider the relations $r_1(P, Q, R)$ and $r_2(R, S, T)$ with primary keys P and R respectively. The relation r_1 contains 2000 tuples and r_2 contains 2500 tuples.*

The maximum size of the join $r_1 \bowtie r_2$ is: ***GATE IT 2006***

A. 2000
B. 2500
C. 4500
D. 5000

GATEPYQ 38 *Consider the relation Student($\underline{name}, sex, marks$), pertaining to students in a class that has at least one boy and one girl.*

What does the following relational algebra expression produce? (Note: ρ is the rename operator.) ***GATE CSE 2004***

$$\pi_{name}(\sigma_{sex=female}(\textit{Student})) \; - \; \pi_{name}\Big(\textit{Student} \bowtie_{(sex=female \wedge x=male \wedge marks \leq m)} \rho_{n,x,m}(\textit{Student})\Big)$$

A. Names of girl students with the highest marks
B. Names of girl students with more marks than some boy student
C. Names of girl students with marks not less than some boy student
D. Names of girl students with more marks than all the boy students

GATEPYQ 39 *Let $R_1(\underline{A}, B, (C))$ and $R_2(\underline{D}, E)$ be two relation schemas, where the primary keys are shown underlined, and let C be a foreign key in R_1 referring to R_2. Suppose there is no violation of the above referential integrity constraint in the corresponding relation instances r_1 and r_2.*

Which one of the following relational algebra expressions would necessarily produce an empty relation? ***GATE CSE 2004***

A. $\Pi_D(r_2) - \Pi_C(r_1)$
B. $\Pi_C(r_1) - \Pi_D(r_2)$
C. $\Pi_D(r_1 \bowtie_{C \neq D} r_2)$
D. $\Pi_C(r_1 \bowtie_{C = D} r_2)$

GATEPYQ 40 *Let r and s be two relations over the relation schemes R and S respectively, and let A be an attribute in R.*

Then the relational algebra expression

$$\sigma_{A=a}(r \bowtie s)$$

is always equal to: ***GATE CSE 2001***

A. $\sigma_{A=a}(r)$
B. $\sigma_{A=a}(r) \bowtie s$
C. $r \bowtie \sigma_{A=a}(s)$
D. None of the above

GATEPYQ 41 *Given the relations*

$$\textit{employee}(\textit{name}, \textit{salary}, \textit{dept-no}), \quad \textit{department}(\textit{dept-no}, \textit{dept-name}, \textit{address})$$

Which of the following queries cannot be expressed using the basic relational algebra operations

$$\sigma, \pi, \times, \bowtie, \cup, \cap, -$$

? ***GATE CSE 2000***

A. Department address of every employee
B. Employees whose name is the same as their department name
C. The sum of all employees' salaries
D. All employees of a given department

GATEPYQ 42 *Consider the join of a relation R with a relation S. If R has m tuples and S has n tuples, then the maximum and minimum sizes of the join respectively are:* ***GATE CSE 1999***

A. $m + n$ and 0
B. mn and 0
C. $m + n$ and $|m - n|$
D. mn and $m + n$

GATEPYQ 43 *Which of the following query transformations (i.e., replacing the l.h.s. expression by the r.h.s. expression) is incorrect?*

R_1 *and* R_2 *are relations,* C_1 *and* C_2 *are selection conditions, and* A_1 *and* A_2 *are attributes of* R_1. ***GATE CSE 1998***

A. $\sigma_{C_1}(\sigma_{C_2}(R_1)) \rightarrow \sigma_{C_2}(\sigma_{C_1}(R_1))$
B. $\sigma_{C_1}(\pi_{A_1}(R_1)) \rightarrow \pi_{A_1}(\sigma_{C_1}(R_1))$
C. $\sigma_{C_1}(R_1 \cup R_2) \rightarrow \sigma_{C_1}(R_1) \cup \sigma_{C_1}(R_2)$
D. $\pi_{A_1}(\sigma_{C_1}(R_1)) \rightarrow \sigma_{C_1}(\pi_{A_1}(R_1))$

GATEPYQ 44 *Given two union compatible relations*

$$R_1(A, B) \quad \textit{and} \quad R_2(C, D)$$

what is the result of the operation

$$R_1 \bowtie_{A=C \wedge B=D} R_2 \ ?$$

GATE CSE 1998

A. $R_1 \cup R_2$
B. $R_1 \times R_2$
C. $R_1 - R_2$
D. $R_1 \cap R_2$

GATEPYQ 45 *Consider two relations describing teams and players in a sports league:*

teams(*tid*, *tname*) : *tid, tname are team-id and team-name, respectively*

players(*pid*, *pname*, *tid*) : *pid, pname, tid denote player-id, player-name and the team-id of the player, respectively*

Which ONE of the following tuple relational calculus queries returns the names of the players who play for the team having name 'MI'? ***GATE CSE 2025***

A. $\{p.pname \mid p \in \textit{players} \wedge \exists t(t \in \textit{teams} \wedge p.tid = t.tid \wedge t.tname =' MI')\}$
B. $\{p.pname \mid p \in \textit{teams} \wedge \exists t(t \in \textit{players} \wedge p.tid = t.tid \wedge t.tname =' MI')\}$
C. $\{p.pname \mid p \in \textit{players} \wedge \exists t(t \in \textit{teams} \wedge t.tname =' MI')\}$
D. $\{p.pname \mid p \in \textit{teams} \wedge \exists t(t \in \textit{players} \wedge t.tname =' MI')\}$

GATEPYQ 46 *Consider a database that has the relation schemas*

EMP(EmpId, EmpName, DeptId), *DEPT(DeptName, DeptId)*

Note that the attribute DeptId can be NULL in the relation EMP. Consider the following queries on the database expressed in tuple relational calculus:

I. $\{t \mid \exists u \in \textit{EMP}(t[\textit{EmpName}] = u[\textit{EmpName}] \wedge \forall v \in \textit{DEPT}(t[\textit{DeptId}] \neq v[\textit{DeptId}]))\}$

II. $\{t \mid \exists u \in \textit{EMP}(t[\textit{EmpName}] = u[\textit{EmpName}] \wedge \exists v \in \textit{DEPT}(t[\textit{DeptId}] \neq v[\textit{DeptId}]))\}$

III. $\{t \mid \exists u \in \textit{EMP}(t[\textit{EmpName}] = u[\textit{EmpName}] \wedge \exists v \in \textit{DEPT}(t[\textit{DeptId}] = v[\textit{DeptId}]))\}$

Which of the above queries are safe? ***GATE CSE 2017***

A. I and II only
B. I and III only
C. II and III only
D. I, II and III

GATEPYQ 47 *The relational algebra expression equivalent to the following tuple relational calculus expression:*

$$\{t \mid t \in r \wedge (t[A] = 10 \wedge t[B] = 20)\}$$

is: ***GATE CSE 1999***

A. $\sigma_{(A=10 \vee B=20)}(r)$
B. $\sigma_{A=10}(r) \ \cup \ \sigma_{B=20}(r)$
C. $\sigma_{A=10}(r) \ \cap \ \sigma_{B=20}(r)$
D. $\sigma_{A=10}(r) \ - \ \sigma_{B=20}(r)$

4.6 Try it Yourself

Exercise 123 *Consider the relations:* ***Employee****(eid, ename, dept),* ***Project****(pid, pname),* ***WorksOn****(eid, pid)*
Which relational algebra expression returns the eids of employees who work on all projects that 'ProjectX' belongs to? (MCQ)
A. $\Pi_{eid}(WorksOn \div \Pi_{pid}(\sigma_{pname='ProjectX'}(Project)))$
B. $\Pi_{eid}(\sigma_{pname='ProjectX'}(Project) \div WorksOn)$
C. $\Pi_{eid}(\sigma_{dept='D1'}(Employee) \bowtie WorksOn)$
D. $\Pi_{eid}(Employee \bowtie WorksOn \bowtie \sigma_{pname='ProjectX'}(Project))$

Exercise 124 *Relations:* ***Student****(sid, sname),* ***Course****(cid, cname),* ***Enroll****(sid, cid)*
The following RA expression:

$$T_1 \leftarrow \Pi_{cid}(\sigma_{sname='Bob'}(Student \bowtie Enroll)), \quad Result \leftarrow Enroll \div T_1$$

What does $Result$ *represent? (MCQ)*
A. Students who have taken all courses that Bob has taken
B. Courses that Bob has not taken
C. Students who have taken at least one course that Bob has taken
D. Students who have never taken the same courses as Bob

Exercise 125 *Consider the relations:* ***Supplier****(sid, sname),* ***Part****(pid, pname, color),* ***Catalog****(sid, pid, cost)*
Which relational algebra expression returns all pairs of suppliers (s1, s2) such that s1 charges more than s2 for some common part? (MSQ)
A. $\Pi_{C1.sid,C2.sid}(\sigma_{C1.pid=C2.pid \wedge C1.cost>C2.cost}(\rho_{C1}(Catalog) \times \rho_{C2}(Catalog)))$
B. $\Pi_{sid,sid}(Catalog \bowtie_{pid=pid \wedge cost>cost} Catalog)$
C. $\Pi_{C1.sid,C2.sid}(\sigma_{C1.pid=C2.pid \wedge C1.cost<C2.cost}(\rho_{C1}(Catalog) \times \rho_{C2}(Catalog)))$
D. $\Pi_{C1.sid,C2.sid}(\sigma_{C1.cost>C2.cost}(\rho_{C1}(Catalog) \times \rho_{C2}(Catalog)))$

Exercise 126 *Relations:* ***Author****(aid, aname),* ***Book****(bid, btitle),* ***AuthorBook****(aid, bid)*
Write a relational algebra expression to find the authors who have written all books titled 'Data Structures'. (NAT)
Answer: $\Pi_{aid}(AuthorBook \div \Pi_{bid}(\sigma_{btitle='DataStructures'}(Book)))$

Exercise 127 *Consider relations:* ***Employee****(eid, ename, dept),* ***Salary****(eid, amount)*
Find the employees earning more than every employee in department 'D2'. (MCQ)
A. $\Pi_{eid}(\sigma_{amount>all(\Pi_{amount}(\sigma_{dept='D2'}(Salary \bowtie Employee)))}(Salary))$
B. $\Pi_{eid}(Salary \bowtie \sigma_{dept='D2'}(Employee))$
C. $\Pi_{eid}(\sigma_{amount>max(\Pi_{amount}(Salary))}(Salary))$
D. $\Pi_{eid}(Salary)$

Exercise 128 *Relations:* ***Student****(sid, sname),* ***Course****(cid, cname),* ***Enroll****(sid, cid)*
How many tuples will result from:

$$\Pi_{sid}(Enroll \bowtie \rho_{sid1,cid1}(Enroll))$$

if there are 10 students and each student is enrolled in 5 courses? (NAT)
Answer: $10 \times 5 \times 5 = 250$

Exercise 129 *Relations:* ***R1****(A, B),* ***R2****(B, C)*
Which relational algebra expression cannot be expressed using only $\sigma, \pi, \times, \bowtie, \cup, \cap, -$*? (MCQ)*
A. $\Pi_A(R1 \bowtie R2)$
B. $\sigma_{B>10}(R1 \bowtie R2)$
C. Sum of C over all tuples of R2
D. $R1 \bowtie R2$

Exercise 130 *Consider relations:* ***Employee****(eid, ename),* ***Dependent****(did, eid, dname)*
Which RA expression returns employees who have at least one dependent older than 18? (MCQ)
A. $\Pi_{eid}(\sigma_{age>18}(Employee \bowtie Dependent))$
B. $\Pi_{eid}(Employee) - \Pi_{eid}(\sigma_{age \leq 18}(Employee \bowtie Dependent))$
C. $\Pi_{eid}(\sigma_{age>18}(Dependent))$
D. $\Pi_{eid}(\sigma_{age>18}(Employee))$

Exercise 131 *Relations:* ***Course****(cid, cname),* ***Prereq****(cid, pre_cid)*
Write a relational algebra expression to find courses that have no prerequisites. (NAT)
Answer: $\Pi_{cid}(Course) - \Pi_{cid}(Prereq)$

Exercise 132 *Relations:* ***Employee****(eid, ename),* ***WorksOn****(eid, pid),* ***Project****(pid, pname)*
Find all employees working on exactly the same set of projects as employee E1. (MSQ)
A. $T1 \leftarrow \Pi_{pid}(\sigma_{ename='E1'}(Employee \bowtie WorksOn))\ T2 \leftarrow \Pi_{eid}(WorksOn \div T1)$
B. $\Pi_{eid}(\sigma_{eid \neq E1}(WorksOn) \bowtie T1)$
C. $\Pi_{eid}(WorksOn \div T1)$
D. $\Pi_{eid}(Employee \bowtie WorksOn)$

Exercise 133 *Consider relations:*

$$\textit{Customer}(cid, cname), \quad \textit{Purchase}(pid, cid, amount)$$

Which TRC query lists all customers who have made a purchase over 1000?
A. $\{c.cname \mid c \in \textit{Customer} \wedge \exists p \in \textit{Purchase}(p.cid = c.cid \wedge p.amount > 1000)\}$
B. $\{c.cname \mid c \in \textit{Customer} \wedge \forall p \in \textit{Purchase}(p.cid = c.cid \wedge p.amount > 1000)\}$
C. $\{c.cname \mid c \in \textit{Customer} \wedge \neg\exists p \in \textit{Purchase}(p.amount \leq 1000)\}$
D. $\{c.cname \mid c \in \textit{Customer}\}$

Exercise 134 *Given relations:*

$$\textit{Employee}(eid, ename, salary), \quad \textit{Project}(pid, eid)$$

Which TRC query returns employees working on all projects?
A. $\{e.ename \mid e \in \textit{Employee} \wedge \forall p \in \textit{Project}(\exists p1 \in \textit{Project}(p1.eid = e.eid))\}$
B. $\{e.ename \mid e \in \textit{Employee} \wedge \exists p \in \textit{Project}(p.eid = e.eid)\}$
C. $\{e.ename \mid e \in \textit{Employee} \wedge \forall p \in \textit{Project}(p.eid \neq e.eid)\}$
D. $\{e.ename \mid e \in \textit{Employee}\}$

Exercise 135 *Consider relations:*

$$\textit{Movie}(mid, mname, director), \quad \textit{Actor}(aid, aname), \quad \textit{ActsIn}(aid, mid)$$

Which TRC query returns actors who acted in all movies directed by 'Spielberg'?
A. $\{a.aname \mid a \in \textit{Actor} \wedge \forall m \in \textit{Movie}(m.director =' Spielberg' \rightarrow \exists x \in \textit{ActsIn}(x.aid = a.aid \wedge x.mid = m.mid))\}$
B. $\{a.aname \mid a \in \textit{Actor} \wedge \exists x \in \textit{ActsIn}(x.aid = a.aid)\}$
C. $\{a.aname \mid a \in \textit{Actor} \wedge \forall x \in \textit{ActsIn}(x.aid = a.aid)\}$
D. $\{a.aname \mid a \in \textit{Actor}\}$

Exercise 136 *Given relations:*

$$\textit{Supplier}(sid, sname), \quad \textit{Parts}(pid, pname), \quad \textit{Supplies}(sid, pid)$$

Which TRC query lists suppliers supplying all parts?
A. $\{s.sname \mid s \in \textit{Supplier} \wedge \forall p \in \textit{Parts}(\exists x \in \textit{Supplies}(x.sid = s.sid \wedge x.pid = p.pid))\}$
B. $\{s.sname \mid s \in \textit{Supplier} \wedge \exists x \in \textit{Supplies}(x.sid = s.sid)\}$
C. $\{s.sname \mid s \in \textit{Supplier} \wedge \forall x \in \textit{Supplies}(x.sid = s.sid)\}$
D. $\{s.sname \mid s \in \textit{Supplier}\}$

Exercise 137 *Consider relations:*

$$\textit{Player}(pid, pname, team), \quad \textit{Team}(tid, tname)$$

Which TRC query lists players who do NOT belong to team 'MI'?
A. $\{p.pname \mid p \in \textit{Player} \wedge \neg\exists t \in \textit{Team}(t.tid = p.team \wedge t.tname =' MI')\}$
B. $\{p.pname \mid p \in \textit{Player} \wedge \exists t \in \textit{Team}(t.tid = p.team \wedge t.tname =' MI')\}$
C. $\{p.pname \mid p \in \textit{Player}\}$
D. $\{p.pname \mid p \in \textit{Team} \wedge t.tname \neq' MI'\}$

4.7 YouTube Links and QR Codes

Lecture	Details	YouTube Link	QR Code
20	Relational Algebra Basics — Selection, Projection, Cartesian Product & Rename — DBMS — Lec. 20	https://youtu.be/lIdn4IEdQHE	
21	Relational Algebra Joins — Theta, Equi, Natural & Outer Joins — DBMS — Lec. 21	https://youtu.be/uckybExR23U	
22	Relational Algebra Set Operations & Division Operation — DBMS — Lec. 22	https://youtu.be/aFLCACDcljs	
23	Relational Algebra Solved Examples — DBMS — Lec. 23	https://youtu.be/MDQLvONNAjY	

24	Problem Solving on Relational Algebra — DBMS — Lec. 24	https://youtu.be/X1j2e0Crrl4	
25	GATE PYQs on Relational Algebra — DBMS — Lec. 25	https://youtu.be/rXzzIc99B1w	
26	Tuple Relational Calculus (TRC) — DBMS — Lec. 26	https://youtu.be/TO33MxTuRqQ	
27	Tuple Relational Calculus (TRC) — Solved GATE PYQs & Practice Problems — DBMS — Lec. 27	https://youtu.be/zexCoq_YBEU	

Chapter 5

Structured Query Language (SQL)

History and Evolution of SQL

SQL (Structured Query Language) was originally developed by IBM in the early 1970s as part of the System R project. It was initially called **SEQUEL** (Structured English Query Language). Over time, the language evolved and became standardized as SQL.
Key milestones in SQL standardization:

- **SQL-86**: The first standard published by ANSI and ISO in 1986.
- **SQL-89**: An extended standard published by ANSI in 1989.
- **SQL-92**: Major revision introducing many features widely used today.
- Subsequent standards: SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016.

SQL is now the **standard relational database language** supported by nearly all modern database systems.

Parts of SQL

SQL is a comprehensive language that can be divided into several functional components:

1. **Data Definition Language (DDL)**: Commands for defining, altering, and deleting database schemas. Examples include `CREATE`, `ALTER`, and `DROP`.
2. **Data Manipulation Language (DML)**: Commands for querying and modifying data, such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.
3. **Integrity**: SQL allows specifying integrity constraints (like `PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`, `CHECK`) to ensure data correctness.
4. **View Definition**: DDL commands to define **views**, which are virtual tables derived from base tables.
5. **Transaction Control Langauge (TCL)**: Commands like `COMMIT`, `ROLLBACK`, and `SAVEPOINT` for managing transactions.
6. **Embedded and Dynamic SQL**: Mechanisms to include SQL statements within general-purpose programming languages like C, C++, and Java.
7. **Authorization**: DDL commands for specifying access rights to relations and views.

SQL Features Overview

- Most database systems implement the SQL-92 standard as a baseline.
- Advanced SQL features, such as triggers, stored procedures, and recursive queries, are often implementation-dependent.
- Always consult the specific database documentation for non-standard features or version differences.

5.1 SQL Schema Creation and Insertion

5.1.1 Basic Data Type

SQL Basic Data Types

SQL provides a variety of built-in data types to store different kinds of values. Each data type has its own characteristics and constraints. The most commonly used types include:

- **char(n)**: A fixed-length character string with a specified length `n`. If the stored string is shorter than `n`, extra spaces are appended to fill the length. Full form: `character(n)`.
- **varchar(n)**: A variable-length character string with a maximum length `n`. Only the actual string is stored, without extra spaces. Full form: `character varying(n)`.
- **int / integer**: Stores integer numbers. The exact range is machine-dependent.
- **smallint**: Stores smaller integer numbers, a subset of `int`.
- **numeric(p,d)**: A fixed-point number with total precision `p` (number of digits) and scale `d` (number of digits to the right of the decimal point). Example: `numeric(3,1)` allows storing 44.5 exactly but not 444.5 or 0.32.
- **real**: Single-precision floating-point number with machine-dependent precision.
- **double precision**: Double-precision floating-point number with higher precision than `real`.
- **float(n)**: Floating-point number with at least `n` digits of precision.
- **nvarchar(n)**: Variable-length Unicode string supporting multilingual characters.

Null values: All SQL data types can include the special `NULL` value, which represents missing or unknown data. Null values indicate that a value may exist but is currently unknown, or that it may not exist at all. Certain columns can be restricted to disallow `NULL`.

Rules and Observations for Character Types

- `char(n)` always stores fixed-length strings. Shorter strings are padded with spaces to reach the specified length.
- `varchar(n)` stores variable-length strings. Only the actual string is stored; no extra spaces are added.
- When comparing two `char` values of different lengths, SQL automatically pads the shorter string with spaces for comparison.
- Comparing `char` and `varchar` may behave differently depending on the database system. Some systems pad the `varchar`, while others do not.
- To avoid unexpected comparison results, it is generally recommended to use `varchar` instead of `char`.
- `nvarchar` (or UTF-8 `varchar`) should be used to store multilingual or Unicode data.

Rules and Observations for Numeric Types

- `int` and `smallint` store integer values. `smallint` uses less storage but has a smaller range.
- `numeric(p,d)` stores fixed-point numbers with exact precision. Values exceeding the specified precision or scale are not allowed.
- `real`, `double precision`, and `float(n)` store approximate floating-point numbers. Floating-point operations may introduce rounding errors.
- When using numeric types, always choose the type that fits the required precision and storage constraints.

Null Values

- `NULL` indicates missing, unknown, or inapplicable data.
- `NULL` is distinct from zero, empty string, or any other default value.
- Columns can be restricted using `NOT NULL` to ensure that data is always provided.
- Operations involving `NULL` generally result in `NULL` (e.g., arithmetic, comparisons), except when explicitly handled.

Best Practices

- Prefer `varchar` over `char` for most text fields to avoid padding issues.
- Use `nvarchar` when storing multilingual data.
- Use `numeric(p,d)` for financial or fixed-point calculations to maintain exact precision.
- Always consider `NULL` constraints and handle missing data carefully.

Example 106: Character Types char and varchar

Consider two attributes:

- Attribute `A` of type `char(10)`
- Attribute `B` of type `varchar(10)`

Example Values:

- Storing "Avi" in `A` → "Avi " (7 trailing spaces added to make length 10)
- Storing "Avi" in `B` → "Avi" (no extra spaces)
- Storing "Samantha" in `A` → "Samantha " (2 trailing spaces)
- Storing "Samantha" in `B` → "Samantha" (length exactly stored)

Observations:

- Comparison of two `char` values automatically pads the shorter string with spaces. For example, "Avi " = "Avi " → TRUE
- Comparison of `char` with `varchar` may fail in some systems: "Avi " = "Avi" → may return FALSE

Example 107: Numeric Types int smallint numeric real float

Integer Examples:

- `int` can store: 0, 10, -500, 100000

- `smallint` can store: 0, 10, -32000, 32000 (smaller range than int)

Fixed-Point Numeric Examples:

- `numeric(5,2)` → 5 total digits, 2 after decimal
  - 123.45 → valid
  - 1.23 → valid (stored as 1.23)
  - 1234.56 → invalid (exceeds 5 digits)
- `numeric(3,1)` → 3 digits, 1 decimal place
  - 44.5 → valid
  - 444.5 → invalid
  - 0.32 → invalid (requires 1 digit after decimal)

Floating-Point Examples:

- `real` or `float(7)`
  - Can store 3.141592, 2.71828, -0.00001
  - Approximate values; may have rounding errors
- `double precision`
  - Higher precision than real
  - Suitable for scientific calculations

Example 108: Null Value Examples

Consider an attribute `MiddleName` of type `varchar(20)`:

- Person1 → "Ravi"
- Person2 → NULL (middle name unknown)
- Person3 → "" (empty string)

Observations:

- NULL is different from an empty string ""
- Arithmetic or string operations with NULL generally result in NULL
- Columns can be constrained with `NOT NULL` to prevent missing values

Example 109: Unicode and Multilingual Strings

Attribute `Language` of type `nvarchar(20)`:

- "English" → stored as usual
- "Español" → stored correctly with ñ

Observations:

- `nvarchar` allows multilingual characters in a single column
- UTF-8 `varchar` may also support Unicode, depending on the database

5.1.2 Basic Schema Definition

Defining SQL Relations

An SQL relation (table) is defined using the `CREATE TABLE` command.
A relation consists of:

- A name (relation name)
- A set of attributes (columns), each with a name and domain (data type)
- Optional integrity constraints (primary key, foreign key, NOT NULL, unique, etc.)

General Syntax:

```
CREATE TABLE relation_name (
    attribute1 datatype [constraint],
    attribute2 datatype [constraint],
    ...,
    [table-level constraints]
);
```

Notes:

- Semicolon at the end of a SQL statement is optional in many implementations.
- Newly created tables are initially empty; data is inserted later.

Integrity Constraints in SQL

Integrity constraints maintain correctness of data in the database.

- **Primary Key:** Attributes that uniquely identify each tuple. Cannot be NULL and must be unique.
- **Foreign Key:** Ensures that attribute values in one table correspond to values of primary key attributes in another table.
- **Not Null:** Prevents NULL values in a column.
- **Composite Keys:** Primary or foreign keys can consist of multiple attributes to uniquely identify a tuple.

Example 110: Department Table

The `department` table stores information about university departments.

- `dept_name`: Name of the department (primary key). Maximum 20 characters.
- `building`: Building of the department. Maximum 15 characters.
- `budget`: Budget of the department. Numeric(12,2).

```
CREATE TABLE department (
    dept_name VARCHAR(20),
    building VARCHAR(15),
    budget NUMERIC(12,2),
    PRIMARY KEY (dept_name)
);
```

Explanation:

- Primary key ensures unique department names.

- Building and budget are optional unless specified NOT NULL.

Example 111: Course Table

The course table stores courses offered by departments.

- course_id: Unique course identifier (primary key), max 7 chars.
- title: Course title, max 50 chars.
- dept_name: Foreign key referencing department.dept_name.
- credits: Numeric(2,0) indicating course credits.

```
CREATE TABLE course (
    course_id VARCHAR(7),
    title VARCHAR(50),
    dept_name VARCHAR(20),
    credits NUMERIC(2,0),
    PRIMARY KEY (course_id),
    FOREIGN KEY (dept_name) REFERENCES department(dept_name)
);
```

Explanation:

- Ensures each course references a valid department.
- Primary key guarantees unique course IDs.

Example 112: Instructor Table

Stores instructor information.

- ID: Instructor ID (primary key), max 5 chars.
- name: Instructor name, max 20 chars, NOT NULL.
- dept_name: Foreign key referencing department.
- salary: Numeric(8,2) salary.

```
CREATE TABLE instructor (
    ID VARCHAR(5),
    name VARCHAR(20) NOT NULL,
    dept_name VARCHAR(20),
    salary NUMERIC(8,2),
    PRIMARY KEY (ID),
    FOREIGN KEY (dept_name) REFERENCES department(dept_name)
);
```

Explanation:

- NOT NULL ensures every instructor has a name.
- Foreign key maintains department validity.

Example 113: Section Table

Stores class sections.

- Composite primary key: `course_id`, `sec_id`, `semester`, `year`.
- `building`, `room_number`, `time_slot_id` store section location and schedule.

```
CREATE TABLE section (
    course_id VARCHAR(8),
    sec_id VARCHAR(8),
    semester VARCHAR(6),
    year NUMERIC(4,0),
    building VARCHAR(15),
    room_number VARCHAR(7),
    time_slot_id VARCHAR(4),
    PRIMARY KEY (course_id, sec_id, semester, year),
    FOREIGN KEY (course_id) REFERENCES course(course_id)
);
```

Explanation:

- Composite key ensures each section is uniquely identified.
- Foreign key ensures section references a valid course.

Example 114: Teaches Table

Records which instructors teach which sections.

- Composite primary key: `ID`, `course_id`, `sec_id`, `semester`, `year`.
- Foreign keys ensure valid instructor IDs and valid section references.

```
CREATE TABLE teaches (
    ID VARCHAR(5),
    course_id VARCHAR(8),
    sec_id VARCHAR(8),
    semester VARCHAR(6),
    year NUMERIC(4,0),
    PRIMARY KEY (ID, course_id, sec_id, semester, year),
    FOREIGN KEY (ID) REFERENCES instructor(ID),
    FOREIGN KEY (course_id, sec_id, semester, year)
        REFERENCES section(course_id, sec_id, semester, year)
);
```

Explanation:

- Composite primary key prevents duplicate teaching assignments.
- Ensures only valid instructors teach valid sections.

5.1.3 Data Insertion

INSERT Statement in SQL

The INSERT statement is used to add tuples (rows) into a relation (table).
There are two common forms:
Form 1 — Insert with attribute list:

```
INSERT INTO table_name (A1, A2, ..., An)
VALUES (v1, v2, ..., vn);
```

Form 2 — Insert without attribute list (full tuple order):

```
INSERT INTO table_name
VALUES (v1, v2, ..., vn);
```

Key Ideas:

- Values must match the attribute data types.
- Order must match attribute order if column list is omitted.
- String values are written in quotes.
- Numeric values are written without quotes.
- Missing attributes get NULL (if allowed).

Rules with Constraints During INSERT

- Primary key values must be **unique and non-null**.
- NOT NULL attributes must be given values.
- Foreign key values must already exist in the referenced table.
- If a foreign key value does not exist in the parent table, insertion fails.
- Data type and size limits are enforced (e.g., VARCHAR length, NUMERIC precision).

Insertion Order with Foreign Keys

When foreign key constraints exist, tables must be populated in dependency order.
Rule: Insert into parent tables first, then child tables.
If table B has a foreign key referencing table A:

- Insert into table A first.
- Insert into table B next.

Otherwise, the DBMS rejects the insert due to referential integrity violation.
Typical Order Example:

1. department (parent)
2. course (references department)
3. section (references course)
4. enrollment (references section)

Example 115: Parent and Child Tables Setup

We create two related tables: `department` and `employee`. Employee has a foreign key referencing department.

```
CREATE TABLE department (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(20),
    budget NUMERIC(10,2)
);

CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(20),
    dept_id INT,
    salary INT,
    FOREIGN KEY (dept_id) REFERENCES department(dept_id)
);
```

Example 116: Insert into Parent Table First

Insert rows into the parent table `department` first.

```
INSERT INTO department VALUES (10, 'Sales', 500000.00);
INSERT INTO department VALUES (20, 'HR', 300000.00);
INSERT INTO department VALUES (30, 'Tech', 900000.00);
```

Department Table After Insert:

dept_id	dept_name	budget
10	Sales	500000.00
20	HR	300000.00
30	Tech	900000.00

Example 117: Insert into Child Table with Valid Foreign Keys

Now insert into the child table `employee`. Each `dept_id` must already exist in department.

```
INSERT INTO employee VALUES (1, 'Asha', 10, 60000);
INSERT INTO employee VALUES (2, 'Ravi', 30, 85000);
INSERT INTO employee VALUES (3, 'Meera', 20, 50000);
```

Employee Table After Insert:

emp_id	emp_name	dept_id	salary
1	Asha	10	60000
2	Ravi	30	85000
3	Meera	20	50000

Example 118: Foreign Key Violation Example

Attempt to insert an employee with a non-existing department.

```
INSERT INTO employee VALUES (4, 'Kiran', 99, 70000);
```

Step-by-step outcome:

- DB checks foreign key: dept_id = 99
- No such dept_id exists in department table
- Insert is rejected
- No row is added

Result: Referential integrity error.

Example 119: Insert Using Column List

We can insert specifying attribute names in any order.

```
INSERT INTO employee (emp_id, emp_name, salary, dept_id)
VALUES (5, 'Neel', 72000, 10);
```

Why useful:

- Order of columns can change
- Some nullable columns may be skipped
- Safer when table has many attributes

Example 120: Multiple Row Insert

Many systems support inserting multiple rows in one statement.

```
INSERT INTO department VALUES
(40, 'Finance', 650000.00),
(50, 'Admin', 200000.00);
```

Department Table After Insert:

dept_id	dept_name	budget
40	Finance	650000
50	Admin	200000

Example 121: Insert Using Arithmetic Expressions

Arithmetic expressions can be used inside VALUES.

```
INSERT INTO employee VALUES (6, 'Pooja', 30, 50000 + 5000);
```

Stored salary becomes: 55000
Explanation: Expression is evaluated first, then inserted.

5.2 SQL Data Definition / Modification Commands

SQL DDL/DML Commands Overview

SQL provides commands to define, modify, and manage database objects and data.
Key Commands:

- **DELETE:** Removes rows from a table based on a condition.
- **UPDATE** : Modifies existing values in table rows.
- **ALTER:** Modifies the structure of an existing table (add/drop/modify columns, constraints).
- **DROP:** Removes entire database objects like tables, views, or indexes.
- **TRUNCATE:** Removes all rows from a table quickly, resetting storage and optionally identity counters.

5.2.1 DELETE

Example 122: DELETE Example

Relation: student(sid, name, age, dept)

```
-- Delete students in 'Physics' department
DELETE FROM student
WHERE dept = 'Physics';
```

Result: Only rows where dept = 'Physics' are removed; table structure remains.

Referential Integrity in DELETE

When a row in a **parent table** is deleted, the DBMS must ensure that no invalid references remain in related tables.
If another table contains a **Foreign Key** referencing the deleted row, the database must decide what to do.

Possible Actions

When a referenced row is deleted, the DBMS can apply one of the following rules:

- **RESTRICT (Default)**
  Deletion is rejected if related rows exist.
  Fails if instructors belong to CS.
- **CASCADE**
  Deletes the referencing rows automatically.
  Deleting department → deletes its instructors.
- **SET NULL**
  Foreign key values in referencing table become NULL.
  Instructor still exists but department becomes unknown.

Specifying Deletion / Update Policies

```
CREATE TABLE instructor (
    ID char(5) PRIMARY KEY,
```

```
    name varchar(20),
    dept_name varchar(20),
    FOREIGN KEY (dept_name)
        REFERENCES department(dept_name)
        ON DELETE CASCADE
        ON UPDATE SET NULL
);
```

Meaning

- **ON DELETE CASCADE** → deleting a department deletes its instructors.
- **ON UPDATE SET NULL** → if department name changes, instructor.dept_name becomes NULL.
- If no rule is specified → **RESTRICT (default)** prevents deletion/update.

Example 123: Deletion Failure Example (RESTRICT)

department

dept_name
CS
Math
Physics

instructor

name	dept_name
Alice	CS
Bob	Math

Query

```
DELETE FROM department
WHERE dept_name = 'CS';
```

Result
Deletion **fails**.
Reason: Instructor **Alice** still references department **CS**.
Database prevents the operation to maintain **referential integrity**.

Example 124: Deletion with CASCADE

Before Deletion
department

dept_name
CS
Math

instructor

name	dept_name
Alice	CS
Bob	Math

Query

```
DELETE FROM department
WHERE dept_name = 'CS';
```

Result (CASCADE)
department

dept_name
Math

instructor

name	dept_name
Bob	Math

Instructor **Alice** is automatically deleted.

5.2.2 UPDATE

UPDATE Statement — Modify Existing Rows

The **UPDATE** command is used to modify existing values in a table.
Syntax

```
UPDATE table_name
SET column1 = value1,
    column2 = value2
WHERE condition;
```

Important Points

- Updates one or more columns in existing rows.
- **WHERE clause** specifies which rows to update.
- If **WHERE is omitted**, all rows in the table are updated.
- Multiple columns can be updated in a single statement.

- Can use expressions or values from other columns.

Example 125: UPDATE Command Example

student

ID	Name	Marks
1	Ravi	70
2	Neha	80
3	Amit	65

Query

```
UPDATE student
SET Marks = 85
WHERE Name = 'Neha';
```

After Update

ID	Name	Marks
1	Ravi	70
2	Neha	85
3	Amit	65

Explanation
Updates the marks of student **Neha** from 80 to 85.

5.2.3 ALTER

ALTER Command

The **ALTER TABLE** command is used to modify the structure of an existing table. It can add new columns, modify column definitions, or remove columns.

Example 126: ALTER Example

Relation: student(sid, name, age)

```
-- Add a new column 'dept'
ALTER TABLE student
ADD dept VARCHAR(20);

-- Modify column 'age' datatype
ALTER TABLE student
MODIFY age INT;

-- Drop column 'dept'
ALTER TABLE student
```

```
DROP COLUMN dept;
```

Result: Table structure is changed; existing data may be affected based on operation.

5.2.4 DROP

DROP Command

The **DROP** command is used to permanently remove a database object such as a table. Both the table structure and all stored data are deleted.

Example 127: DROP Example

Relation: student(sid, name, age, dept)

```
-- Drop the entire table
DROP TABLE student;
```

Result: Table and all its data are permanently deleted.

5.2.5 TRUNCATE

TRUNCATE Command

The **TRUNCATE** command removes all rows from a table quickly. The table structure remains unchanged.

Example 128: TRUNCATE Example

Relation: student(sid, name, age, dept)

```
-- Remove all rows quickly
TRUNCATE TABLE student;
```

Result: Table structure remains, but all rows are removed. Identity columns reset (if any).

5.3 SQL Query and Basic Operations

Basic Structure of SQL Queries

An SQL query consists of three main clauses:

- **SELECT:** Specifies the attributes or expressions to appear in the result.
- **FROM:** Specifies the relations (tables) from which tuples are retrieved.
- **WHERE:** (Optional) Specifies conditions to filter tuples.

General Syntax:

```
SELECT attribute1, attribute2, ...
FROM relation1 [, relation2, ...]
WHERE condition;
```

Notes:

- Queries produce a relation as output.

- By default, SQL allows duplicates. Use `DISTINCT` to remove duplicates.
- Logical operators (`AND`, `OR`, `NOT`) and comparison operators (`=`, `<>`, `>`, `>=`, `<`, `<=`) can be used in the WHERE clause.
- Arithmetic expressions (`+`, `-`, `*`, `/`) can be used in the SELECT clause.

Rules and Properties

- `DISTINCT` removes duplicates from the result.
- `ALL` explicitly keeps duplicates (default behavior).
- Arithmetic operations in SELECT compute values on-the-fly; they do not change table data.
- Logical connectives allow combining multiple conditions.
- WHERE filters rows based on comparison or logical expressions.

5.3.1 Select all Attributes (*)

Concept

The asterisk symbol (*) in the SELECT clause denotes **all attributes**.
Two important forms exist:
1. SELECT *

```
SELECT *
FROM table_list;
```

Returns all attributes of the **result relation** formed by the FROM clause.
2. table_name.*

```
SELECT table_name.*
FROM table_list;
```

Returns all attributes of only that specific table.
Properties:

- Column order follows schema definition order.
- Includes all columns — even if later added.
- Common in quick inspection queries.
- Not recommended in production views (schema-change risk).

Concept

When multiple tables appear in FROM:

- SELECT * $\rightarrow$ returns all columns from all tables
- SELECT A.* $\rightarrow$ only columns from table A
- SELECT A.*, B.col $\rightarrow$ mix of full table + specific columns
- Duplicate column names may appear in result

If duplicate attribute names exist, qualify them using table or alias.

Example 129: SELECT * from Single Table

student

sid	name	age
S1	Anil	20
S2	Meera	21

Query:

```
SELECT *
FROM student;
```

Result Relation:

sid	name	age
S1	Anil	20
S2	Meera	21

5.3.2 Renaming in SQL (AS)

Attribute and Relation Renaming in SQL

SQL provides the **AS clause** to rename attributes (columns) and relations (tables) within the result of a query. This is called the **rename operation**.
Renaming is required because:

- Different relations may contain attributes with the same name.
- Arithmetic expressions in SELECT do not have default column names.
- Derived columns need meaningful labels.
- Long table names should be shortened for readability.
- Self-joins require multiple logical copies of the same table.

Renaming does not change the base table — it only affects the query result.

AS Clause — Syntax Rules

AS can be used in two places.
Column rename (SELECT clause):

```
SELECT column_or_expression AS new_name
FROM table;
```

Relation rename (FROM clause — table alias):

```
SELECT ...
FROM table_name AS alias_name;
```

Rules and properties:

- AS keyword is optional in many DBMS but recommended.
- Alias exists only during query execution.

- After aliasing, use alias.column instead of table.column.
- Alias is mandatory in self-join queries.
- Alias improves readability and reduces typing.

5.3.3 Relational Operators: Comparison Operators

Concept

Concept of Relational Operators
Relational operators (also known as **Comparison Operators**) are used in SQL to compare two expressions. When a comparison is made, the result is a **Boolean value**: either `TRUE`, `FALSE`, or `UNKNOWN` (in the case of `NULL` values). These operators are the backbone of data filtering in the `WHERE` clause.
The Standard Relational Operators are:

- = : Equal to
- <> or != : Not equal to
- > : Greater than
- < : Less than
- >= : Greater than or equal to
- <= : Less than or equal to

Example 130: Using Greater Than or Equal To ($\geq$)

Consider the following **employee** table as our dataset:
Employee:

Name	Department	Salary	JoinDate
Alice	Sales	70000	2022-05-12
Bob	Engineering	65000	2023-03-10
Carol	Marketing	80000	2022-12-01
David	Sales	55000	2021-11-30
Eve	Sales	70000	2024-02-15
Frank	HR	52000	2023-01-20

Find employees earning 70000 or more:

```
SELECT Name, Salary
FROM employee
WHERE Salary >= 70000;
```

Output Relation:

Name	Salary
Alice	70000
Carol	80000
Eve	70000

Example 131: Using Not Equal To ($\neq$)

Find all employees who are NOT in the 'Sales' department:

```
SELECT Name, Department
FROM employee
WHERE Department <> 'Sales';
```

Note: <> and != are both accepted in most SQL dialects.

Output Relation:

Name	Department
Bob	Engineering
Carol	Marketing
Frank	HR

Example 132: Using Less Than ($<$)

Find employees hired before the year 2023:

```
SELECT Name, JoinDate
FROM employee
WHERE JoinDate < '2023-01-01';
```

Output Relation:

Name	JoinDate
Alice	2022-05-12
Carol	2022-12-01
David	2021-11-30

5.3.4 Arithmetic Operations

Arithmetic Operators in SQL SELECT

SQL supports arithmetic operations in the SELECT clause:

- + addition
- − subtraction

- * multiplication
- / division

These operations can be applied to numeric attributes or constants.

```
SELECT salary * 1.1 AS new_salary, salary + 500 AS incremented_salary
FROM employee;
```

Explanation: The query shows salary after 10% increase and after adding 500.

Example 133: Employee Table

Input table employee:

ID	Name	Salary
E01	Alice	70000
E02	Bob	65000
E03	Carol	80000
E04	Dave	55000
E05	Eve	90000

Example 134: Select Names of All Employees

```
SELECT Name
FROM employee;
```

Output Relation:

Name
Alice
Bob
Carol
Dave
Eve

Example 135: Select Salary with Arithmetic Expressions

```
SELECT ID, Name, Salary, Salary * 1.1 AS SalaryAfterRaise, Salary + 2000 AS Bonus
FROM employee;
```

Output Relation:

ID	Name	Salary	SalaryAfterRaise	Bonus
E01	Alice	70000	77000	72000
E02	Bob	65000	71500	67000
E03	Carol	80000	88000	82000
E04	Dave	65000	60500	57000
E05	Eve	90000	99000	92000

Example 136: Filter Rows Using WHERE Clause

Find employees with salary $>$ 70000:

```
SELECT Name, Salary
FROM employee
WHERE Salary > 70000;
```

Output Relation:

Name	Salary
Carol	80000
Eve	90000

5.3.5 Basic Logical Operators: AND OR

Example 137: Using Logical Operators in WHERE Clause

Find employees earning more than 60000 and less than 90000:

```
SELECT Name, Salary
FROM employee
WHERE Salary > 60000 AND Salary < 90000;
```

Output Relation:

Name	Salary
Alice	70000
Bob	65000
Carol	80000

Example 138: Using DISTINCT to Remove Duplicates

Suppose an employee table with duplicate salaries:

```
SELECT DISTINCT Salary
FROM employee;
```

Output Relation:

Salary
55000
65000
70000
80000
90000

5.3.6 String Operations

Concept

SQL represents strings using **single quotes**.
Examples of string literals:

- 'Computer'
- 'Database Systems'
- 'CS101'

If a single quote appears inside a string, it must be written twice.
Example:

- Text: It's correct
- SQL string: 'It''s correct'

String comparison using equality ($=$) is defined as **case sensitive** in the SQL standard. Thus:

- 'data' $=$ 'DATA' $\rightarrow$ false (standard behavior)

However, some systems (like MySQL default collations) may treat string comparison as case-insensitive unless configured otherwise.

Concept

SQL supports many string operations. Exact names may vary slightly across DBMS.
Common ones:
Concatenation

- Standard operator: `||`
- MySQL alternative: `CONCAT(s1,s2)`

Case conversion

- `UPPER(s)` — convert to uppercase
- `LOWER(s)` — convert to lowercase

Length

- `LENGTH(s)` — number of characters

Substring

- `SUBSTRING(s, start, len)`

Trim spaces

- `TRIM(s)` — removes leading/trailing spaces

Properties:

- Functions can appear in SELECT and WHERE.
- Function results can be renamed using AS.
- Functions operate row-by-row.

Concept

SQL supports pattern matching using the **LIKE** operator.
Special pattern characters:

- % → matches any substring (zero or more characters)
- _ → matches exactly one character

Pattern matching is case sensitive in standard SQL, but may be case-insensitive in some systems.
Forms:

```
column LIKE pattern
column NOT LIKE pattern
```

Examples of patterns:

- 'Data
- '
- '___' → exactly 3 characters
- '__

Concept

If pattern must include special characters (% or _), use ESCAPE.
Syntax:

```
column LIKE pattern ESCAPE 'escape_char'
```

Rules:

- Escape character appears before special symbol.
- That symbol is treated as normal text.

Example meanings:

- LIKE 'ab%%' ESCAPE '
- LIKE 'file_1' ESCAPE '

Example 139: String Equality and Quotes

Table: course

course_id	title
C1	Data Science
C2	database systems
C3	Data Mining

Find exact match.

```
SELECT course_id
FROM course
WHERE title = 'Data Science';
```

Step-by-step:

- Compare full string
- Case matters (standard SQL)
- Only exact match selected

Result:

course_id
C1

Example 140: Concatenation of Strings

Table: student

first_name	last_name
Ravi	Kumar
Neha	Shah

Create full name.

```
SELECT first_name || ' ' || last_name AS full_name
FROM student;
```

Result:

full_name
Ravi Kumar
Neha Shah

Example 141: UPPER

Table: department

dept_name
Physics
CompSci
Math

```
SELECT dept_name,
       UPPER(dept_name) AS upper_name,
       LOWER(dept_name) AS lower_name,
       LENGTH(dept_name) AS len
FROM department;
```

Result:

dept_name	upper_name	lower_name	len
Physics	PHYSICS	physics	7
CompSci	COMPSCI	compsci	7
Math	MATH	math	4

Example 142: LIKE with Percent Pattern

Table: building

building
Watson Hall
New Watson Block
Taylor Center

Find names containing Watson.

```
SELECT building
FROM building
WHERE building LIKE '%Watson%';
```

Result:

building
Watson Hall
New Watson Block

Example 143: LIKE with Underscore Pattern

Table: codes

code
A12
B9X
AB12
C123

Match exactly 3-character codes.

```
SELECT code
FROM codes
WHERE code LIKE '___';
```

Result:

code
A12
B9X

Example 144: LIKE with ESCAPE Character

Table: files

fname
data%report
data_set
data100

Match literal percent sign.

```
SELECT fname
FROM files
WHERE fname LIKE 'data\%%' ESCAPE '\';
```

Step-by-step:

- % treated as normal character
- Then wildcard

Result:

fname
data%report

Example 145: Selecting All Attributes with *

Table: trainer

```
SELECT *
FROM trainer;
```

Meaning:

- Returns all attributes of the result relation.
- Equivalent to listing every column explicitly.

Qualified form:

```
SELECT T.*
FROM trainer AS T;
```

Returns all columns only from alias T.

5.4 ORDER BY

ORDER BY Clause — Sorted Output

SQL provides the **ORDER BY** clause to control the order in which tuples (rows) are displayed in the query result. Basic syntax:

```
SELECT attributes
FROM table_list
WHERE condition
ORDER BY attribute_list;
```

Key properties:

- Sorting is applied to the final result relation.
- Default order is **ascending**.
- Sorting can be done on one or more attributes.
- Does not change stored table order — only display order.
- Can be used with expressions and aliases.

Concept

Sort direction can be controlled explicitly.

```
ORDER BY column ASC    -- ascending (default)
ORDER BY column DESC   -- descending
```

Multi-attribute sorting is evaluated left to right:

- First attribute sorted fully
- Ties are broken using next attribute
- Each attribute may have its own ASC/DESC

Example 146: Ordering by One Attribute — Alphabetic

instructor

ID	name	dept
I3	Kiran	Physics
I1	Anil	Physics
I2	Meera	Chemistry

Query:

```
SELECT name
FROM instructor
WHERE dept = 'Physics'
ORDER BY name;
```

Result Relation (ascending by name):

name
Anil
Kiran

Example 147: Descending Order

employee

name	salary
Ravi	60000
Neha	90000
Asha	75000

Query:

```
SELECT name, salary
FROM employee
ORDER BY salary DESC;
```

Result Relation (highest first):

name	salary
Neha	90000
Asha	75000
Ravi	60000

Example 148: Multi-Attribute Ordering

instructor

name	salary	dept
Riya	80000	CS
Amit	90000	CS
Neha	90000	EE
Karan	80000	ME

Requirement: Sort by salary descending, then name ascending.

```
SELECT *
FROM instructor
ORDER BY salary DESC, name ASC;
```

Result Relation:

name	salary	dept
Amit	90000	CS
Neha	90000	EE
Karan	80000	ME
Riya	80000	CS

Tie handling:

- Salary sorted first (DESC)
- Equal salary → name sorted ASC

Example 149: Ordering Using Expression

sales

item	qty	price
Pen	10	5
Book	4	120
Bag	2	900

Order by computed total value.

```
SELECT item, qty * price AS total_value
FROM sales
ORDER BY total_value DESC;
```

Result Relation:

item	total_value
Bag	1800
Book	480
Pen	50

Example 150: Ordering with Alias Name

Alias can be used inside ORDER BY.

```
SELECT name, salary AS pay
FROM instructor
ORDER BY pay DESC;
```

Sorting uses the alias column label.

5.5 BETWEEN

Concept

SQL provides the **BETWEEN** operator to simplify range conditions in the WHERE clause.
Instead of writing two comparisons with AND, BETWEEN expresses a closed interval test.
Syntax:

```
column BETWEEN low_value AND high_value
```

Equivalent form:

```
column >= low_value AND column <= high_value
```

Key properties:

- Range is **inclusive** of both limits.
- Works with numeric, date, and string types.
- Improves readability of range predicates.
- Can be negated using NOT BETWEEN.

Concept

NOT BETWEEN selects values outside the range.

```
column NOT BETWEEN a AND b
```

SQL also supports **row constructors** — tuple values written as:

```
(v1, v2, ..., vn)
```

Tuple comparisons are allowed and use **lexicographic ordering**.
For tuples:

- (a1, a2) = (b1, b2) → both components equal
- (a1, a2) ¡ (b1, b2) → compared left to right

- Used for compact multi-column comparisons

Example 151: BETWEEN — Salary Range Query

instructor

name	salary
Ravi	88000
Neha	92000
Amit	97000
Kiran	105000

Query:

```
SELECT name
FROM instructor
WHERE salary BETWEEN 90000 AND 100000;
```

Result Relation:

name
Neha
Amit

Equivalent condition:

```
WHERE salary >= 90000 AND salary <= 100000;
```

Example 152: NOT BETWEEN Example

Query instructors outside the mid salary band.

```
SELECT name
FROM instructor
WHERE salary NOT BETWEEN 90000 AND 100000;
```

Result:

name
Ravi
Kiran

Example 153: BETWEEN on Strings

BETWEEN also works on strings using alphabetical order.
department

dept_name
Biology
Chemistry
Mathematics
Physics

Query:

```
SELECT dept_name
FROM department
WHERE dept_name BETWEEN 'C' AND 'P';
```

Result (alphabetic range):

dept_name
Chemistry
Mathematics
Physics

Example 154: Row Constructor — Multi-Column Equality

instructor

ID	name	dept
I1	Rao	Biology
I2	Sen	Physics

teaches

ID	course
I1	BIO101
I2	PHY201

Tuple comparison query:

```
SELECT name, course
FROM instructor, teaches
WHERE (instructor.ID, dept) =
      (teaches.ID, 'Biology');
```

Result Relation:

name	course
Rao	BIO101

Example 155: Lexicographic Tuple Comparison

Tuple comparisons follow left-to-right ordering.

```
SELECT *
FROM instructor
WHERE (salary, name) >= (90000, 'M');
```

Interpretation:

- First compare salary
- If salary equal → compare name
- Acts like dictionary ordering

5.6 NULL Values

What is a NULL Value?

NULL represents a missing, unknown, or inapplicable value.
It does **not** mean:

- zero
- empty string
- false

NULL means: *value exists but is not known or not provided.*

Effect of NULL in Different Operations

1. Arithmetic Operations
Any arithmetic expression involving NULL evaluates to NULL.

```
A + 5      NULL if A is NULL
A * 10      NULL if A is NULL
A / B      NULL if A or B is NULL
```

Reason: Result cannot be determined if an input is unknown.

2. Comparison Operations
Any comparison with NULL results in **UNKNOWN**.

```
salary > 5000          UNKNOWN if salary is NULL
A = NULL               UNKNOWN
1 < NULL               UNKNOWN
NULL = NULL            UNKNOWN
```

SQL uses **three-valued logic**: TRUE, FALSE, UNKNOWN.

Three-Valued Logic with NULL

Boolean operators are extended to handle UNKNOWN.
AND rules

- TRUE AND UNKNOWN = UNKNOWN
- FALSE AND UNKNOWN = FALSE

- UNKNOWN AND UNKNOWN = UNKNOWN

OR rules

- TRUE OR UNKNOWN = TRUE
- FALSE OR UNKNOWN = UNKNOWN
- UNKNOWN OR UNKNOWN = UNKNOWN

NOT rule

- NOT UNKNOWN = UNKNOWN

If a WHERE predicate evaluates to FALSE or UNKNOWN, the tuple is **not selected**.

Example 156: Arithmetic with NULL

employee

id	bonus
E1	500
E2	NULL

```
SELECT id, bonus + 100
FROM employee;
```

Result:
E1 → 600
E2 → NULL

Example 157: Comparison with NULL in WHERE

```
SELECT id
FROM employee
WHERE bonus > 200;
```

If bonus is NULL → condition becomes UNKNOWN → row excluded.

Testing for NULL Correct Method

NULL cannot be tested using = or ¡¿.
Wrong:

```
WHERE salary = NULL
```

Correct:

```
WHERE salary IS NULL
WHERE salary IS NOT NULL
```

Example 158: Find Rows with NULL Salary

```
SELECT name
FROM instructor
```

```
WHERE salary IS NULL;
```

Testing UNKNOWN Result

SQL allows explicit test of UNKNOWN predicate result.

```
condition IS UNKNOWN
condition IS NOT UNKNOWN
```

Example 159: UNKNOWN Predicate Check

```
SELECT name
FROM instructor
WHERE (salary > 10000) IS UNKNOWN;
```

Selects rows where salary is NULL.

NULL and DISTINCT

In duplicate elimination (DISTINCT):
Two NULL values are treated as **equal**.
Example tuples:

$$(A, NULL), (A, NULL)$$

DISTINCT keeps only one copy.
Note: This differs from predicate logic where

```
NULL = NULL      UNKNOWN
```

NULL in Set Operations

For UNION, INTERSECT, EXCEPT:
Tuples are considered identical if:

- All non-NULL attributes equal
- Corresponding attributes are both NULL

Therefore duplicate NULL tuples collapse into one in set operations.

Example 160: DISTINCT with NULL

table T

A	B
1	NULL
1	NULL

```
SELECT DISTINCT * FROM T;
```

Result keeps only one tuple.

5.7 AGGREGATE FUNCTIONS

Aggregate Functions

Aggregate Functions: These operations process a collection of values (either a set or a multiset) to produce a single scalar output.

SQL Aggregate Functions

Standard SQL Aggregates: SQL provides five primary built-in functions:

- `avg`: Calculates the arithmetic mean.
- `min`: Identifies the minimum value.
- `max`: Identifies the maximum value.
- `sum`: Computes the numerical total.
- `count`: Returns the number of elements in the collection.

Data Type Constraints: While `sum` and `avg` are strictly limited to numeric datasets, `min`, `max`, and `count` are versatile enough to handle non-numeric data, such as character strings.

Aggregate Function — Basic Syntax

```
SELECT AGG_FUNCTION(column)
FROM table
WHERE condition;
```

Examples:

```
SELECT AVG(salary) FROM instructor;
SELECT SUM(credits) FROM course;
SELECT MAX(age) FROM student;
```

COUNT Variants

COUNT(*)

- Counts every row
- Includes rows containing NULL

COUNT(column)

- Counts only rows where column is NOT NULL

COUNT(DISTINCT column)

- Counts distinct non-NULL values only

5.7.1 Aggregation and Null Values

NULL Values and Aggregate Functions — Core Rule

In SQL aggregation, NULL values are treated specially.
Rule: All aggregate functions **ignore NULL values** in their input, except COUNT(*).

This prevents unknown values from corrupting aggregate results.

Behavior of Each Aggregate with NULL

Assume relation instructor(name, dept, salary) where some salary values are NULL.
SUM, AVG, MIN, MAX

- NULL values are skipped
- Only non-NULL values are used

COUNT(column)

- Counts only non-NULL values

COUNT(*)

- Counts all rows
- NULL does not matter

Example 161: Aggregate Ignoring NULL

instructor

name	salary
A	50000
B	NULL
C	70000

Query:

```
SELECT SUM(salary), AVG(salary), COUNT(salary), COUNT(*)
FROM instructor;
```

Computed on values: 50000, 70000
Result:

- SUM = 120000
- AVG = 60000
- COUNT(salary) = 2
- COUNT(*) = 3

Row with NULL salary is ignored except in COUNT(*).

Aggregates on Empty Input Set

After NULL removal, the value set may become empty.
SQL defines:

- COUNT → returns 0
- SUM → returns NULL
- AVG → returns NULL
- MIN → returns NULL

- MAX → returns NULL

Reason: Only COUNT has a well-defined identity for empty sets.

Example 162: Empty Aggregate Case

```
SELECT AVG(salary)
FROM instructor
WHERE dept = 'Astronomy';
```

If no matching rows → result is NULL (not 0).

Example 163: Aggregate Example Table

instructor

name	salary
A	50000
B	60000
C	NULL
D	60000

Example 164: AVG and SUM Example

```
SELECT AVG(salary), SUM(salary)
FROM instructor;
```

NULL salary is ignored in computation.
AVG = (50000 + 60000 + 60000)/3
SUM = 170000

Example 165: MIN and MAX Example

```
SELECT MIN(salary), MAX(salary)
FROM instructor;
```

Result:
MIN = 50000
MAX = 60000

Example 166: COUNT Difference Example

```
SELECT COUNT(*), COUNT(salary)
FROM instructor;
```

COUNT(*) = 4
COUNT(salary) = 3

DISTINCT with Aggregates

DISTINCT can be used inside aggregate functions.

```
SELECT COUNT(DISTINCT salary)
FROM instructor;
```

Counts only unique non-NULL values.

5.8 Groups

5.8.1 GROUP BY HAVING

GROUP BY — Core Concept

GROUP BY is used to divide tuples into groups based on one or more attributes, and then apply aggregate functions on each group separately.
Instead of one aggregate result for the whole table, we get one aggregate result per group.
Typical use: summaries per department, per course, per year, etc.

Logical Query Execution Order with GROUP BY

SQL evaluates grouped queries logically in this order:

1. FROM
2. WHERE
3. GROUP BY
4. AGGREGATE
5. HAVING
6. SELECT
7. ORDER BY

Meaning:

- WHERE filters rows first
- GROUP BY forms groups
- Aggregates are computed per group
- HAVING filters groups
- SELECT outputs final columns

GROUP BY — Basic Syntax

```
SELECT group_column, AGG_FUNC(column)
FROM table
WHERE row_condition
GROUP BY group_column;
```

Each distinct value of group_column forms one group.

GROUP BY — Mandatory Rule

In SELECT list of a grouped query:
Every attribute must be either:

- present in GROUP BY, OR
- used inside an aggregate function

Otherwise SQL raises an error.
Invalid:

```
SELECT dept, salary
FROM instructor
GROUP BY dept;
```

salary is neither grouped nor aggregated.

HAVING — Syntax

```
SELECT group_col, AGG_FUNC(col)
FROM table
WHERE row_condition
GROUP BY group_col
HAVING aggregate_condition;
```

HAVING condition is applied per group.

Example 167: HAVING — Departments with Avg Salary > 70000

```
SELECT dept, AVG(salary)
FROM instructor
GROUP BY dept
HAVING AVG(salary) > 70000;
```

Evaluation:
CS avg = 85000 → kept
EE avg = 65000 → removed

GROUP BY HAVING — Logical Processing Steps

For a query:

```
SELECT dept, AVG(salary)
FROM instructor
WHERE salary > 50000
GROUP BY dept
HAVING AVG(salary) > 70000;
```

SQL processes it in this exact order:

1. FROM — load tables
2. WHERE — filter rows
3. GROUP BY — create groups
4. Aggregate — compute per group

5. HAVING — filter groups
6. SELECT — produce output columns
7. ORDER BY — sort result (if present)

Example 168: Step-by-Step GROUP BY Execution

instructor

name	dept	salary
A	CS	80000
B	CS	90000
C	EE	70000
D	EE	40000
E	ME	60000

Query:

```
SELECT dept, AVG(salary)
FROM instructor
WHERE salary >= 60000
GROUP BY dept;
```

Step 1 — FROM
All rows are read.
Step 2 — WHERE filter
Keep only salary 60000
Remaining rows:
A CS 80000
B CS 90000
C EE 70000
E ME 60000
Row D removed.
Step 3 — GROUP Formation
Rows are partitioned by dept:
CS → (80000, 90000)
EE → (70000)
ME → (60000)
Step 4 — Aggregate Computation
AVG per group:
CS → 85000
EE → 70000
ME → 60000
Step 5 — SELECT Output
Final relation:

dept	avg
CS	85000
EE	70000
ME	60000

Example 169: Adding HAVING

Query:

```
SELECT dept, AVG(salary)
FROM instructor
WHERE salary >= 60000
GROUP BY dept
HAVING AVG(salary) > 65000;
```

Steps 1–4 same as before.
Step 5 — HAVING filter on groups
CS → 85000 → keep
EE → 70000 → keep
ME → 60000 → remove
Final Output
CS, EE only.

Example 170: GROUP BY with COUNT

```
SELECT dept, COUNT(*)
FROM instructor
GROUP BY dept;
```

Counts total rows per department (NULL allowed).

Grouping by Multiple Attributes

GROUP BY can use multiple attributes.
Each unique combination forms a group.

```
GROUP BY dept, year
```

Grouping key = (dept, year)

Example 171: Multi-Attribute Grouping

```
SELECT dept, salary, COUNT(*)
FROM instructor
GROUP BY dept, salary;
```

Each (dept, salary) pair becomes one group.

HAVING — Group Filter

HAVING filters groups after aggregation.
WHERE filters rows before grouping.
Key difference:

- WHERE → cannot use aggregates
- HAVING → can use aggregates

Example 172: WHERE + HAVING Together

```
SELECT dept, AVG(salary)
FROM instructor
WHERE salary > 60000
GROUP BY dept
HAVING AVG(salary) > 75000;
```

Steps:
WHERE → remove rows salary 60000
GROUP → form groups
AVG → compute per group
HAVING → filter groups

Example 173: HAVING with COUNT

Find departments with at least 2 instructors.

```
SELECT dept, COUNT(*)
FROM instructor
GROUP BY dept
HAVING COUNT(*) >= 2;
```

HAVING Without GROUP BY

If GROUP BY is absent, entire table is treated as one group.
HAVING can still be used.

```
SELECT AVG(salary)
FROM instructor
HAVING AVG(salary) > 70000;
```

Example 174: GROUP BY + ORDER BY

```
SELECT dept, AVG(salary) AS avg_sal
FROM instructor
GROUP BY dept
ORDER BY avg_sal DESC;
```

ORDER BY applies after grouping.

Common GROUP BY Errors

- Selecting non-grouped, non-aggregated column
- Using aggregate in WHERE
- Forgetting that WHERE runs before grouping
- Confusing HAVING with WHERE

Rule: Row filter → WHERE
Group filter → HAVING

5.8.2 GROUP BY and Null Values

GROUP BY with NULL Values

When grouping:

- All NULL values form one group
- Aggregates inside the group still ignore NULL inputs
- Unlike comparisons, GROUP BY treats NULL = NULL for grouping purposes.

So:

```
GROUP BY dept
```

All rows with dept = NULL go into one group.

Trap Points

- SUM ignores NULL — does not return NULL unless no values remain
- COUNT(column) ≠ COUNT(*)
- AVG never divides by total rows — divides by non-NULL rows only
- Empty aggregate → NULL (except COUNT)
- NULL group exists in GROUP BY

5.9 Joins

5.9.1 Queries on Multiple Tables

Concept

Many practical SQL queries need data from more than one relation. Such queries are called **multi-relation queries**. SQL allows combining multiple tables using the `FROM` clause and specifying matching conditions in the `WHERE` clause (or using JOIN syntax).
Core idea:

- Each relation listed in the FROM clause participates in query evaluation.
- SQL conceptually forms a Cartesian product of those relations.
- The WHERE clause filters only meaningful tuple combinations.
- The SELECT clause finally projects required attributes.

Typical business questions answered using multi-relation queries:

- Get employee names with department buildings.
- List students with course titles.
- Show orders with customer details.
- Find instructors with courses taught.

General Form — Multi-Relation Query

General structure:

$$\texttt{SELECT } A_1, A_2, \ldots, A_n \texttt{ FROM } r_1, r_2, \ldots, r_m \texttt{ WHERE } P;$$

Rules:

- FROM lists all relations involved.
- WHERE contains join predicates and filters.
- If attribute names repeat across tables, prefix with relation name.
- If WHERE is omitted → full Cartesian product is returned.
- Logical evaluation order: FROM → WHERE → SELECT.

5.9.2 Cartesian Product or CROSS JOIN

Cartesian Product Concept

If multiple tables are listed in FROM without a matching condition, SQL generates a **Cartesian product**:

- Every tuple of table A pairs with every tuple of table B.
- Result size $= |A| \times |B|$
- Usually very large and rarely meaningful by itself.
- **Cartesian Product Without WHERE Clause is called CROSS JOIN**
- Join predicates in WHERE restrict combinations.

Attribute Name Disambiguation

When the same attribute appears in multiple relations:

- Use qualified names: table.attribute
- Example: `employee.dept_id`, `department.dept_id`
- If attribute appears in only one table → prefix optional
- All relations in FROM must have distinct names (or aliases).

Example 175: Employee–Department Join

Assume two colorful relations:
employee

emp_id	name	dept_code	salary
E1	Arjun	D10	52000
E2	Meera	D20	61000
E3	Kabir	D10	58000
E4	Zara	D30	47000

department

dept_code	dept_name	building
D10	Computing	Block-A
D20	Finance	Block-C
D30	Design	Block-B

Goal: Get employee names with department name and building.
Step-by-step logic:

- FROM combines employee × department
- WHERE keeps rows where dept codes match
- SELECT outputs chosen columns

```
SELECT name, department.dept_name, building
FROM employee, department
WHERE employee.dept_code = department.dept_code;
```

Result Relation

name	dept_name	building
Arjun	Computing	Block-A
Meera	Finance	Block-C
Kabir	Computing	Block-A
Zara	Design	Block-B

Example 176: Cartesian Product Without WHERE - CROSS JOIN

Using same tables but NO matching predicate. **Cartesian Product Without WHERE Clause is called CROSS JOIN**

```
SELECT name, dept_name
FROM employee, department;
```

Step-by-step:

- employee has 4 rows
- department has 3 rows

- Result size $= 4 \times 3 = 12$ rows
- Every employee paired with every department

(Only partial output shown)

name	dept_name
Arjun	Computing
Arjun	Finance
Arjun	Design
Meera	Computing
Meera	Finance
...	...

This is usually incorrect for real queries — missing join condition.

Example 177: Join with Additional Filter + Arithmetic Operator

Find employees with department info whose salary after 10% raise exceeds 60000.

```
SELECT name,
       department.dept_name,
       salary,
       salary * 1.10 AS revised_salary
FROM employee, department
WHERE employee.dept_code = department.dept_code
  AND salary * 1.10 > 60000;
```

Arithmetic operator used: multiplication (*)
Evaluation:

- Join first
- Compute salary * 1.10
- Apply filter condition
- Output projected columns

name	dept_name	salary	revised_salary
Meera	Finance	61000	67100
Kabir	Computing	58000	63800

Example 178: Three-Relation Query

Add third table:
project

dept_code	project_title
D10	AI System
D20	Audit Tool
D30	UX Revamp

Get employee, department, and project.

```
SELECT name, dept_name, project_title
FROM employee, department, project
WHERE employee.dept_code = department.dept_code
  AND department.dept_code = project.dept_code;
```

Process:

- Cartesian product of 3 tables
- Two join predicates restrict matches
- Final projection returned

Rename Tables

Correlation Names / Table Aliases

When a table is renamed in a query using AS, the new name is called a:

- correlation name
- table alias
- tuple variable

All mean the same concept: a temporary query-level name for a relation.

Example 179: Renaming Output Columns

Tables:

trainer

tid	name	skill
T1	Riya	SQL
T2	Dev	Python
T3	Karan	ML

teaches

tid	course_code
T1	DB101
T2	AI201
T1	DB202

Rename output attribute names.

```
SELECT name AS trainer_name,
       course_code AS course_id
FROM trainer, teaches
WHERE trainer.tid = teaches.tid;
```

Result Relation

trainer_name	course_id
Riya	DB101
Dev	AI201
Riya	DB202

Step-by-step:

- Join on matching tid
- Select two attributes
- Rename them using AS

Example 180: Renaming Expression Columns

Arithmetic expressions produce unnamed columns — alias is useful.

trainer

name	hours	rate
Riya	30	500
Dev	25	600

Compute payment.

```
SELECT name,
       hours * rate AS total_payment
FROM trainer;
```

Result Relation

name	total_payment
Riya	15000
Dev	15000

Without AS, the column header would be system-generated.

Example 181: Table Alias for Short Names

Rename tables to simplify query writing.

```
SELECT T.name, C.course_code
FROM trainer AS T, teaches AS C
WHERE T.tid = C.tid;
```

Why alias helps:

- Avoids long prefixes
- Improves readability
- Reduces typing errors
- Clearer join conditions

Example 182: Alias Without Using AS Keyword

Many DBMS allow alias without AS.

```
SELECT name trainer_name
FROM trainer;
```

Equivalent to using AS — but AS is clearer and preferred in textbooks.

5.9.3 SELF JOIN

Example 183: Self Join Using Aliases

Find trainers whose rate is greater than at least one trainer with skill = 'SQL'.

trainer

name	skill	rate
Riya	SQL	500
Dev	Python	650
Karan	SQL	550
Neha	ML	700

Self-join requires aliases to distinguish copies.

```
SELECT DISTINCT A.name
FROM trainer AS A, trainer AS B
WHERE A.rate > B.rate
  AND B.skill = 'SQL';
```

OR

```
SELECT DISTINCT A.name
FROM trainer AS A
JOIN trainer AS B ON A.rate > B.rate
WHERE B.skill = 'SQL';
```

Step-by-step:

- Table used twice → A and B
- Compare rates across rows
- Filter B rows where skill = SQL
- Select A names

Result Relation

name
Dev
Neha
Karan

5.9.4 INNER JOIN

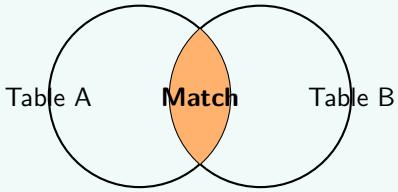
Concept: The Inner Join

The **INNER JOIN** is the most fundamental join type in SQL. It selects records that have matching values in both tables being joined.

Key Characteristics:

- It returns only the **intersection** of two tables.
- If a row in Table A does not have a corresponding match in Table B, it will not appear in the result.
- It is used to combine related data stored in separate tables.

Venn Diagram Representation:



Example 184: Basic Inner Join

Employee Table:

EmpID	Name	DeptID
101	Alice	1
102	Bob	2
103	Carol	1
104	David	4

Department Table:

DeptID	DeptName
1	Sales
2	Engineering
3	Marketing

Find the department name for every employee that has a valid department:

```
SELECT E.Name, D.DeptName
FROM employee AS E
INNER JOIN department AS D ON E.DeptID = D.DeptID;
```

Note: David (DeptID 4) is excluded because 4 does not exist in the Department table. 'Marketing' (DeptID 3) is excluded because no employee belongs to it.

Output Relation:

Name	DeptName
Alice	Sales
Bob	Engineering
Carol	Sales

Example 185: Inner Join with Condition

Find employees specifically in the 'Sales' department:

```
SELECT E.Name, D.DeptName
FROM employee AS E
INNER JOIN department AS D ON E.DeptID = D.DeptID
WHERE D.DeptName = 'Sales';
```

Output Relation:

Name	DeptName
Alice	Sales
Carol	Sales

5.9.5 NATURAL JOIN

Concept: The Natural Join

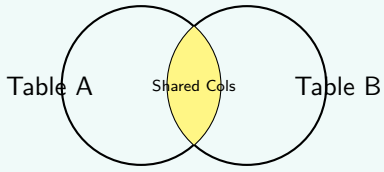
A **NATURAL JOIN** is a specialized type of join that automatically joins two tables based on **all columns** that have the same name and data type in both tables.

Key Characteristics:

- **Implicit Join Condition:** You do not use the `ON` or `USING` clause; SQL identifies matching columns automatically.
- **Column Deduplication:** Unlike a standard Join, the Natural Join automatically removes duplicate columns from the output.

- **Risk Factor:** If tables share common names like `ID` or `Created_At` that aren't intended for joining, it can lead to empty or incorrect results.

Venn Diagram Representation: Like an Inner Join, it represents the intersection, but specifically restricted by shared column names.



Example 186: Natural Join between Students and Courses

In this example, both tables share the column `CourseID`.

Input Tables:

students Table:

Name	CourseID
Alice	101
Bob	102
Carol	101

courses Table:

CourseID	Title
101	SQL
102	Python
103	Java

```
SELECT * FROM students NATURAL JOIN courses;
```

Output Relation:

CourseID	Name	Title
101	Alice	SQL
102	Bob	Python
101	Carol	SQL

Example 187: Natural Join for Orders

Tables share the `ProductID` column.

Input Tables:

orders Table:

OrderID	ProductID
501	P1
502	P2

products Table:

ProductID	Price
P1	25.00
P2	40.00

```
SELECT OrderID, ProductID, Price
FROM orders NATURAL JOIN products;
```

Output Relation:

OrderID	ProductID	Price
501	P1	25.00
502	P2	40.00

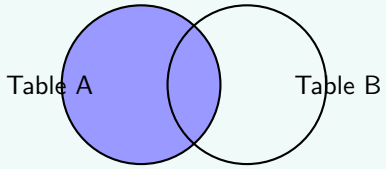
5.9.6 LEFT (OUTER) JOIN

Concept: The LEFT (OUTER) JOIN

The **LEFT JOIN** (or **LEFT OUTER JOIN**) returns **all** records from the left table (Table A), and the matched records from the right table (Table B).
Key Characteristics:

- **Persistence:** Every row from the left table is guaranteed to appear at least once in the result.
- **Handling Mismatches:** If there is no match in the right table for a specific row, the result will contain `NULL` values for all columns of the right table.
- **Directional:** The order in which you list the tables matters significantly.

Venn Diagram Representation: The result is the entirety of Table A, including the overlapping section with Table B.



Example 188: Left Join: Employees and Departments

Identify all employees, including those who have not yet been assigned to a department.
Input Tables:

employee Table (Left):

Name	DeptID
Alice	10
Bob	20
Charlie	*NULL*

department Table (Right):

DeptID	DeptName
10	HR
20	IT

```
SELECT E.Name, D.DeptName
FROM employee E
LEFT JOIN department D ON E.DeptID = D.DeptID;
```

Output Relation:

Name	DeptName
Alice	HR
Bob	IT
Charlie	*NULL*

Example 189: Left Join: Customers and Orders

List all customers and their order amounts. Customers without orders will show `NULL`.
Input Tables:

customers Table (Left):

CustID	Name
C1	John
C2	Sarah

orders Table (Right):

CustID	Amount
C1	250.00

```
SELECT C.Name, O.Amount
FROM customers C
LEFT JOIN orders O ON C.CustID = O.CustID;
```

Output Relation:

Name	Amount
John	250.00
Sarah	*NULL*

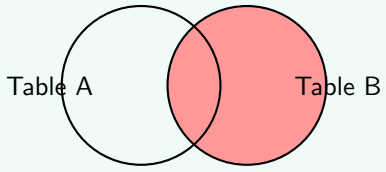
5.9.7 RIGHT (OUTER) JOIN

Concept: The RIGHT (OUTER) JOIN

The **RIGHT JOIN** (or **RIGHT OUTER JOIN**) is the mirror image of the Left Join. It returns **all** records from the right table (Table B), and the matched records from the left table (Table A).
Key Characteristics:

- **Persistence:** Every row from the right table is guaranteed to appear in the result set.
- **Handling Mismatches:** If there is no matching row in the left table for a specific row in the right table, the result will contain `NULL` values for the columns belonging to the left table.
- **Logic:** It is essentially a Left Join with the table order reversed.

Venn Diagram Representation: The result covers the entirety of Table B, including the overlapping section where records match Table A.



Example 190: Right Join: Employees and Departments

Identify all departments, including those that currently have no employees assigned to them.

Input Tables:

employee Table (Left):

Name	DeptID
Alice	10
Bob	10

department Table (Right):

DeptID	DeptName
10	HR
20	Research

```
SELECT E.Name, D.DeptName
FROM employee E
RIGHT JOIN department D ON E.DeptID = D.DeptID;
```

Output Relation:

Name	DeptName
Alice	HR
Bob	HR
NULL	Research

Example 191: Right Join: Books and Authors

List all authors and the titles of books they have written in the database.

Input Tables:

books Table (Left):

Title	AuthorID
SQL Basics	A1

authors Table (Right):

AuthorID	AuthorName
A1	J. Doe
A2	S. Smith

```
SELECT B.Title, A.AuthorName
FROM books B
RIGHT JOIN authors A ON B.AuthorID = A.AuthorID;
```

Output Relation:

Title	AuthorName
SQL Basics	J. Doe
NULL	S. Smith

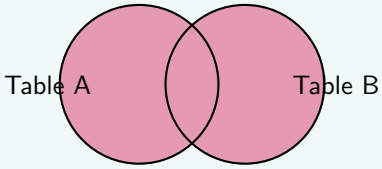
5.9.8 FULL (OUTER) JOIN

Concept: The FULL (OUTER) JOIN

The **FULL JOIN** returns all records when there is a match in either left (Table A) or right (Table B) table records. It essentially combines the results of both a Left and a Right Join.
Key Characteristics:

- **Complete Visibility:** It retains every row from both tables.
- **NULL Padding:** For rows where there is no match, `NULL` values are placed in the columns of the table that lacks the matching record.
- **Compatibility Note:** Some databases (like MySQL) do not support `FULL JOIN` directly and require a `UNION` of a Left and Right Join.

Venn Diagram Representation: The result is the union of both sets, covering all areas of both circles.



Example 192: Full Join: Projects and Departments

List all projects and all departments, showing links where they exist.
Input Tables:

projects Table (Left):

ProjName	DeptID
Alpha	10
Beta	99

departments Table (Right):

DeptID	Name
10	Sales
20	R&D

```
SELECT P.ProjName, D.Name
FROM projects P
FULL OUTER JOIN departments D ON P.DeptID = D.DeptID;
```

Output Relation:

ProjName	Name
Alpha	Sales
Beta	*NULL*
NULL	R&D

5.10 Set Operations

Set Operations

In SQL, the UNION, INTERSECT, and EXCEPT operators allow us to perform set-based logic directly on the results of our queries.

5.10.1 UNION

UNION Operation — Set Combination

The **UNION** operator combines the results of two SELECT queries and returns tuples that appear in either result.
Syntax:

```
SELECT column_list
FROM table1
WHERE condition
UNION
SELECT column_list
FROM table2
WHERE condition;
```

Rules:

- Both queries must return the **same number of columns**.
- Corresponding columns must have **compatible domains**.
- Column names in output come from the first query.
- **Duplicates are automatically removed**.
- Parentheses around subqueries are optional (DB dependent).

Concept

UNION ALL performs multiset union.

```
query1
UNION ALL
query2;
```

Properties:

- Does **not remove duplicates**.
- Faster than UNION (no duplicate elimination step).
- Duplicate count in result = sum of occurrences in both queries.
- Used when duplicate tuples are meaningful.

Example 193: UNION — Courses Offered in Two Semesters

section

course_id	semester	year
CS101	Fall	2017
CS315	Fall	2017
CS101	Spring	2018
CS319	Spring	2018
CS319	Spring	2018

Query:

```
SELECT course_id
FROM section
WHERE semester = 'Fall' AND year = 2017
UNION
SELECT course_id
FROM section
WHERE semester = 'Spring' AND year = 2018;
```

Result Relation (duplicates removed):

course_id
CS101
CS315
CS319

Example 194: UNION ALL — Keeping Duplicates

Using same table.

```
SELECT course_id
FROM section
WHERE semester = 'Fall' AND year = 2017
UNION ALL
SELECT course_id
FROM section
WHERE semester = 'Spring' AND year = 2018;
```

Result Relation (duplicates preserved):

course_id
CS101
CS315
CS101
CS319
CS319

Example 195: UNION with Character Data

ug_students

name
Anil
Meera
Ravi

pg_students

name
Ravi
Kiran

Query:

```
SELECT name FROM ug_students
UNION
SELECT name FROM pg_students;
```

Result:

name
Anil
Meera
Ravi
Kiran

Example 196: UNION Requires Compatible Columns

Valid UNION (same column count and type):

```
SELECT dept_name
FROM department
UNION
SELECT building
FROM department;
```

Invalid UNION (different column counts):

```
-- ERROR
SELECT name, salary FROM instructor
UNION
SELECT name FROM employee;
```

Example 197: UNION with ORDER BY

ORDER BY applies to the **final combined result**.

```
SELECT name FROM ug_students
```

```
UNION
SELECT name FROM pg_students
ORDER BY name;
```

Sorting happens after duplicate elimination.

5.10.2 INTERSECTION

Concept

The **INTERSECT** operator returns only those tuples that appear in **both** query results.
Syntax:

```
SELECT column_list
FROM table1
WHERE condition
INTERSECT
SELECT column_list
FROM table2
WHERE condition;
```

Rules:

- Both queries must return same number of columns.
- Corresponding columns must have compatible domains.
- Result column names come from first query.
- **Duplicates are automatically removed**.
- Output = set intersection of the two results.

Concept

INTERSECT ALL keeps duplicates.

```
query1
INTERSECT ALL
query2;
```

Duplicate rule:

- Duplicate count in result = minimum(count in query1, count in query2)
- Uses multiset (bag) semantics.
- Not supported in many DBMS implementations.

Note:

- **MySQL does not support INTERSECT directly.**
- Workaround uses subqueries with IN or EXISTS.

Example 198: INTERSECT — Courses in Both Semesters

section

course_id	semester	year
CS101	Fall	2017
CS315	Fall	2017
CS101	Spring	2018
CS319	Spring	2018
CS319	Spring	2018

Query:

```
SELECT course_id
FROM section
WHERE semester = 'Fall' AND year = 2017
INTERSECT
SELECT course_id
FROM section
WHERE semester = 'Spring' AND year = 2018;
```

Result Relation (only common course):

course_id
CS101

Example 199: INTERSECT ALL — Duplicate Counting

Assume:

- Fall 2017 has 4 sections of ECE101
- Spring 2018 has 2 sections of ECE101

```
SELECT course_id
FROM section
WHERE semester='Fall' AND year=2017
INTERSECT ALL
SELECT course_id
FROM section
WHERE semester='Spring' AND year=2018;
```

Result contains:

- ECE101 appears **2 times**
- Minimum of (4, 2)

Example 200: INTERSECT with Student Lists

clubA

sid
S1
S2
S3

clubB

sid
S2
S3
S4

```
SELECT sid FROM clubA
INTERSECT
SELECT sid FROM clubB;
```

Result:

sid
S2
S3

Example 201: MySQL Workaround Using IN

Equivalent of INTERSECT in MySQL:

```
SELECT DISTINCT course_id
FROM section
WHERE semester='Fall' AND year=2017
AND course_id IN (
  SELECT course_id
  FROM section
  WHERE semester='Spring' AND year=2018
);
```

Example 202: MySQL Workaround Using EXISTS

```
SELECT DISTINCT s1.course_id
FROM section s1
WHERE semester='Fall' AND year=2017
AND EXISTS (
  SELECT 1
  FROM section s2
  WHERE s2.course_id = s1.course_id
  AND semester='Spring' AND year=2018
);
```

5.10.3 EXCEPT

Concept

The **EXCEPT** operator returns all tuples that appear in the **first query result but not in the second**.
It performs **set difference**.
Syntax:

```
SELECT column_list
FROM table1
WHERE condition
EXCEPT
SELECT column_list
FROM table2
WHERE condition;
```

Rules:

- Both queries must return same number of columns.
- Corresponding columns must have compatible domains.
- Result column names come from first query.
- Duplicate tuples are removed before difference.
- Output = tuples in query1 tuples in query2.

Concept

EXCEPT ALL keeps duplicate counts.

```
query1
EXCEPT ALL
query2;
```

Duplicate rule:

- Result count = count(query1) count(query2)
- Included only if difference is positive.
- Uses multiset (bag) semantics.

DBMS notes:

- Oracle uses keyword **MINUS** instead of EXCEPT.
- MySQL does **not support EXCEPT**.
- MySQL workaround uses NOT IN / NOT EXISTS.

Example 203: EXCEPT — Courses Only in Fall

section

course_id	semester	year
CS101	Fall	2017
CS315	Fall	2017
CS347	Fall	2017
CS101	Spring	2018
CS319	Spring	2018

Query:

```
SELECT course_id
FROM section
WHERE semester='Fall' AND year=2017
EXCEPT
SELECT course_id
FROM section
WHERE semester='Spring' AND year=2018;
```

Result Relation:

course_id
CS315
CS347

CS101 removed because it appears in Spring also.

Example 204: EXCEPT — Duplicate Elimination

Assume:

- Fall has 4 sections of ECE101
- Spring has 2 sections of ECE101

```
SELECT course_id
FROM section
WHERE semester='Fall'
EXCEPT
SELECT course_id
FROM section
WHERE semester='Spring';
```

Result:

- No ECE101 appears
- Because duplicates are removed before subtraction

Example 205: EXCEPT ALL — Duplicate Arithmetic

Using same counts:

```
SELECT course_id
FROM section
WHERE semester='Fall'
EXCEPT ALL
SELECT course_id
FROM section
WHERE semester='Spring';
```

Result:

- ECE101 appears 2 times
- Because 4 2 = 2

Example 206: Student Difference Example

registered

sid
S1
S2
S3
S4

paid_fees

sid
S2
S4

```
SELECT sid FROM registered
EXCEPT
SELECT sid FROM paid_fees;
```

Result:

sid
S1
S3

Example 207: MySQL Workaround Using NOT IN

```
SELECT DISTINCT course_id
FROM section
WHERE semester='Fall' AND year=2017
AND course_id NOT IN (
```

```
  SELECT course_id
  FROM section
  WHERE semester='Spring' AND year=2018
);
```

Example 208: MySQL Workaround Using NOT EXISTS

```
SELECT DISTINCT s1.course_id
FROM section s1
WHERE semester='Fall' AND year=2017
AND NOT EXISTS (
  SELECT 1
  FROM section s2
  WHERE s2.course_id = s1.course_id
  AND semester='Spring' AND year=2018
);
```

5.11 Nested Queries

Nested queries (subqueries) are SQL queries written inside another query. They are evaluated either:

- once (non-correlated subquery), or
- once per outer row (correlated subquery)

They are commonly used with:

- IN / NOT IN
- SOME / ANY / ALL
- EXISTS / NOT EXISTS

5.11.1 Set Membership: IN & NOT IN

IN Set Membership Test

Checks whether a value belongs to the result set of a subquery.
Syntax:

```
value IN (subquery)
```

True if value matches any element returned by subquery.

Example 209: IN Subquery Example

Find instructors who belong to departments located in Delhi.

```
SELECT name
FROM instructor
WHERE dept_name IN (
    SELECT dept_name
    FROM department
    WHERE city = 'Delhi'
);
```

Execution Steps

1. Inner query returns set of dept_name (Delhi departments)
2. Outer query checks membership in that set

Concept

```
value NOT IN (subquery)
```

Equivalent to: value $\neq$ every element of set.
If subquery result contains even one NULL → comparison becomes UNKNOWN → no rows returned.

Example 210: NOT IN with NULL Problem

If subquery returns: {CS, EE, NULL}
Then:

```
dept_name NOT IN (...)
```

Result → UNKNOWN → filtered out.

5.11.2 Set Comparison: SOME, ANY & ALL

Comparison with Subquery Results

Used when subquery returns numeric or comparable values.
Form:

```
value > ANY (subquery)
value > ALL (subquery)
```

Concept

ANY and SOME are synonyms.

```
x > ANY(S)
```

True if x is greater than at least one value in S.
Equivalent to OR over comparisons.

Example 211: ANY Example

Find instructors earning more than at least one CS instructor.

```
SELECT name
FROM instructor
WHERE salary > ANY (
    SELECT salary
    FROM instructor
    WHERE dept_name = 'CS'
);
```

Concept

```
x > ALL(S)
```

True if x is greater than every value in S.
Equivalent to AND over comparisons.

Example 212: ALL Example

Find instructors earning more than all CS instructors.

```
SELECT name
FROM instructor
WHERE salary > ALL (
    SELECT salary
    FROM instructor
    WHERE dept_name = 'CS'
);
```

Equivalent logic: salary $>$ MAX(CS salaries)

Empty Set Behavior

If subquery returns empty set:

- x $>$ ALL(empty) $\rightarrow$ TRUE
- x $>$ ANY(empty) $\rightarrow$ FALSE

5.11.3 Test Empty: EXISTS & NOT EXISTS

EXISTS Tuple Existence Test

Checks whether subquery returns at least one row.

```
EXISTS (subquery)
```

Returns TRUE if subquery result is non-empty.

Example 213: EXISTS Example — Correlated

Find departments that have at least one instructor.

```
SELECT d.dept_name
FROM department d
WHERE EXISTS (
    SELECT *
    FROM instructor i
    WHERE i.dept_name = d.dept_name
);
```

Subquery runs once per department row.

Concept

NOT EXISTS is often used to express "for all" conditions.
Pattern:

```
NOT EXISTS (violating rows)
```

Meaning: no counterexample exists.

Example 214: NOT EXISTS — For All Pattern

Find instructors who teach all courses in CS dept.

```
SELECT i.ID
FROM instructor i
WHERE NOT EXISTS (
    SELECT c.course_id
    FROM course c
    WHERE c.dept_name = 'CS'
    AND NOT EXISTS (
        SELECT *
        FROM teaches t
        WHERE t.ID = i.ID
        AND t.course_id = c.course_id
    )
);
```

Logic: No CS course exists that this instructor does not teach.

Nested Subquery Operators — Outcome Truth Table

Operator	Test Performed	Predicate TRUE When	Empty Subquery	If NULL Present in Subquery
IN	Membership	match exists	FALSE	Safe: if a match exists, TRUE dominates (NULL does not matter)
NOT IN	Non-membership	no match exists	TRUE	NULL trap: becomes UNKNOWN $\rightarrow$ row rejected
$>$ ANY / SOME	Compare with at least one	any comparison TRUE	FALSE	Safe if one TRUE exists (OR logic)
$>$ ALL	Compare with every value	all comparisons TRUE	TRUE	NULL may cause UNKNOWN $\rightarrow$ row rejected
$=$ ANY	Equal to at least one	match exists	FALSE	Same behavior as IN (OR logic)
$=$ ALL	Equal to every value	all values equal x	TRUE	NULL may cause UNKNOWN $\rightarrow$ row rejected
EXISTS	Row existence	subquery returns ≥ 1 row	FALSE	TRUE even if rows contain NULL
NOT EXISTS	No row existence	subquery returns 0 rows	TRUE	FALSE if any row exists

5.11.4 Subqueries in the FROM Clause

Subquery in FROM Clause — Derived Tables

A subquery can appear in the **FROM clause**. Its result is treated as a temporary table (called a **derived table**). Syntax:

```
SELECT ...
FROM (subquery) AS alias
WHERE condition;
```

Rules:

- Alias is **mandatory**
- Subquery is evaluated first
- Outer query runs on the derived result
- Columns of subquery become columns of derived table

Execution order: Subquery $\rightarrow$ temporary table $\rightarrow$ outer query

Example 215: Average Salary Per Department — Two-Level Query

Find departments whose average salary is greater than 70000.

```
SELECT dept_name, avg_sal
FROM (
    SELECT dept_name, AVG(salary) AS avg_sal
    FROM instructor
    GROUP BY dept_name
) AS D
WHERE avg_sal > 70000;
```

Steps:

1. Inner query computes avg salary per department
2. Result becomes derived table D
3. Outer query filters using avg_sal

Example 216: Aggregation then Join

Find departments whose average salary is greater than the overall average salary.

```
SELECT D.dept_name
FROM (
    SELECT dept_name, AVG(salary) AS avg_sal
    FROM instructor
    GROUP BY dept_name
) AS D
WHERE D.avg_sal >
      (SELECT AVG(salary) FROM instructor);
```

Derived table stores group results → compared with scalar subquery.

Example 217: Top Earners Per Department — Derived Table Join

```
SELECT i.name, i.dept_name
FROM instructor i,
     (
       SELECT dept_name, MAX(salary) AS max_sal
       FROM instructor
       GROUP BY dept_name
     ) AS M
WHERE i.dept_name = M.dept_name
AND i.salary = M.max_sal;
```

Pattern: Group → compute aggregate → join back to base table.

5.11.5 The WITH Clause (Common Table Expressions)

WITH Clause — Common Table Expression (CTE)

WITH defines a named temporary relation used inside a query.
Also called: **CTE — Common Table Expression**
Syntax:

```
WITH name AS (subquery)
SELECT ...
FROM name;
```

Properties:

- Improves readability
- Avoids repeating subqueries
- Acts like temporary view
- Exists only for that query
- Can define multiple CTEs

Example 218: WITH instead of FROM Subquery

Same as derived-table example, using WITH.

```
WITH dept_avg AS (
    SELECT dept_name, AVG(salary) AS avg_sal
    FROM instructor
    GROUP BY dept_name
)
SELECT dept_name
FROM dept_avg
WHERE avg_sal > 70000;
```

Cleaner than nested FROM subquery.

Example 219: Multiple CTE Definitions

```
WITH
dept_avg AS (
  SELECT dept_name, AVG(salary) AS a
  FROM instructor
  GROUP BY dept_name
),
overall AS (
  SELECT AVG(salary) AS g
  FROM instructor
)
SELECT dept_name
FROM dept_avg, overall
WHERE a > g;
```

Two temporary tables created → used in final query.

Example 220: Reuse of CTE

```
WITH high_paid AS (
    SELECT *
    FROM instructor
    WHERE salary > 80000
)
```

```sql
SELECT COUNT(*)
FROM high_paid;

SELECT AVG(salary)
FROM high_paid;
```

CTE logic written once → reused multiple times.

5.12 Problems

Problem 138 *Student(rollNo, name, deptId), Grades(rollNo, courseId, grade)*

```
SELECT name
FROM Student
WHERE rollNo IN (
    SELECT rollNo
    FROM Grades
    WHERE grade = 'A'
);
```

What does this query return?
A. Names of students with at least one 'A' grade
B. Names of students with all 'A' grades
C. Names of students with no 'A' grades
D. None of the above

Problem 139 *Employee(empId, name, salary, deptId), Department(deptId, deptName, budget)*

```
SELECT name
FROM Employee e
WHERE EXISTS (
    SELECT *
    FROM Department d
    WHERE e.deptId = d.deptId AND d.budget > 100000
);
```

What does this query return?
A. Employees in departments with budget > 100,000
B. Employees with salary > 100,000
C. All employees
D. None of the above

Problem 140 *Consider the table Employee:*

empId	name	deptId	salary
1	Alice	10	75000
2	Bob	20	60000
3	Charlie	10	NULL
4	David	20	90000
5	Eve	10	70000
6	Frank	20	50000
7	Gim	NULL	80000

Consider the query:

```
SELECT deptId
FROM Employee
GROUP BY deptId
HAVING AVG(salary) > 70000;
```

The number of tuples in the result of above query is ____ (NAT)

Problem 141 *Consider the tables:*
`Student(rollNo, name)`, `Grades(rollNo, courseId, grade)`

rollNo	name
1	Alice
2	Bob
3	Charlie

rollNo	courseId	grade
1	C1	A
1	C2	B
2	C1	B
2	C2	B
3	C1	C
3	C2	A

Query:

```
SELECT name
FROM Student
WHERE rollNo NOT IN (
    SELECT rollNo FROM Grades WHERE grade = 'B'
);
```

What does this query return?
A. Alice, Charlie
B. Bob
C. Alice, Bob
D. Charlie

Problem 142 *Consider the table* `Orders`*:*

orderId	custId	amount
101	1	5000
102	2	7000
103	1	9000
104	3	4000
105	2	6000
106	3	8000

Consider the SQL query:

```
SELECT COUNT(*)
FROM Orders O1
WHERE amount > (
    SELECT AVG(amount)
    FROM Orders O2
    WHERE O1.custId = O2.custId
);
```

What is the number of orders returned by the query?
Answer: ____ *(NAT)*

Problem 143 *Consider the table* `Orders`*:*

orderId	custId	amount
101	1	5000
102	2	7000
103	1	9000
104	3	4000
105	2	6000
106	3	8000

Query:

```
SELECT COUNT(*)
FROM Orders
WHERE custId NOT IN (
    SELECT custId
    FROM Orders
    WHERE amount > 8000
);
```

What is the numeric answer?
Answer: ____ *(NAT)*

Problem 144 *Consider* `Product(productId, price)`*:*

productId	price
1	500
2	700
3	600
4	800
5	900

Query:

```
SELECT COUNT(*)
FROM Product
WHERE price > ANY (
    SELECT price
    FROM Product
    WHERE productId IN (1,2)
);
```

Find the numeric answer.
Answer: ____ *(NAT)*

Problem 145 *Consider* `Product(productId, price)`*:*

productId	*price*
1	*500*
2	*700*
3	*600*
4	*800*
5	*900*

Query:

```
SELECT COUNT(*)
FROM Product
WHERE price > ALL (
    SELECT price
    FROM Product
    WHERE productId IN (1,2)
);
```

How many products satisfy this condition?
Answer: ____ *(NAT)*

Problem 146 *Consider the table* `Employee(empId, salary)`*:*

empId	*salary*
1	*50000*
2	*60000*
3	*75000*
4	*80000*
5	*90000*

Query:

```
SELECT COUNT(*)
FROM Employee
WHERE salary >= SOME (
```

```
    SELECT salary
    FROM Employee
    WHERE empId IN (2,3)
);
```

Find the numeric answer.
Answer: ____ *(NAT)*

Problem 147 *Consider the table* `Sales(saleId, empId, amount)`*:*

saleId	empId	amount
1	1	5000
2	2	7000
3	1	9000
4	2	4000
5	3	8000
6	3	8500

Query:

```
SELECT COUNT(*)
FROM Sales S1
WHERE amount = (
    SELECT MAX(amount)
    FROM Sales S2
    WHERE S1.empId = S2.empId
);
```

How many rows are returned?
Answer: ____ *(NAT)*

Problem 148 ***[NAT]*** *Consider a relation $R(A, B)$ containing n tuples where all values of A and B are distinct and non-null. Let $n = 10$. The following SQL query is executed:*

```
SELECT COUNT(*)
FROM R R1
WHERE R1.A > ANY (SELECT R2.B FROM R R2 WHERE R1.A = R2.A);
```

If for every tuple $(a, b) \in R$, the condition $a > b$ holds true, the number of rows returned by the query is ____.

Problem 149 ***[MCQ]*** *Consider the relations $R(A, B)$ and $S(B, C)$ with the following instances:*
$R = \{(1, 10), (2, 20), (3, NULL)\}$ *and* $S = \{(10, 100), (NULL, 200), (30, 300)\}$.
What is the result of the following query?

```
SELECT COUNT(*)
FROM R LEFT OUTER JOIN S ON R.B = S.B
WHERE S.C IS NULL;
```

A. 1
B. 2
C. 3
D. 0

Problem 150 ***[MSQ]*** *Consider the schema $E(id, name, sal, dno)$ and $D(dno, dname)$. Which of the following queries correctly find the names of departments that have at least one employee, but no employee with a salary strictly greater than 50,000?*

A.

```
SELECT dname FROM D WHERE dno IN (SELECT dno FROM E)
EXCEPT
SELECT dname FROM D WHERE dno IN (SELECT dno FROM E WHERE sal > 50000)
```

B.

```
SELECT dname FROM D JOIN E USING (dno)
GROUP BY dname HAVING MAX(sal) <= 50000
```

C.

```
SELECT dname FROM D d WHERE EXISTS (SELECT * FROM E e WHERE e.dno = d.dno)
AND 50000 >= ALL (SELECT sal FROM E e WHERE e.dno = d.dno)
```

D.

```
SELECT dname FROM D d WHERE NOT EXISTS
(SELECT * FROM E e WHERE e.dno = d.dno AND e.sal <= 50000)
```

Problem 151 ***[NAT]*** *Consider the table `Numbers(val)` with values $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.*

```
SELECT COUNT(*) FROM Numbers n1
WHERE val > (SELECT AVG(val) FROM Numbers n2 WHERE n2.val <= n1.val);
```

The number of rows returned is ____.

Problem 152 ***[MCQ]*** *Consider three relations $R(A)$, $S(B)$, and $T(C)$. R has 5 tuples, S has 10 tuples, and T has 0 tuples. All attributes are of type integer. How many tuples are returned by the following query?*

```
SELECT * FROM R
WHERE A NOT IN (SELECT B FROM S WHERE B IN (SELECT C FROM T));
```

A. 0
B. 5
C. 10
D. Undefined

Problem 153 ***[NAT]*** *Consider the following tables representing a graph: `Nodes(id)` and `Edges(src, dest)`. If `Nodes` contains $\{1, 2, 3, 4, 5\}$ and `Edges` contains $\{(1, 2), (2, 3), (3, 1), (4, 5)\}$, what is the output of:*

```
SELECT COUNT(*) FROM Nodes n
WHERE NOT EXISTS (
    SELECT * FROM Edges e1 WHERE e1.src = n.id
    EXCEPT
    SELECT * FROM Edges e2 WHERE e2.dest = n.id
);
```

Problem 154 ***[MCQ]*** *Consider the relation $P(x, y)$ with 100 tuples. We execute:*

```
DELETE FROM P WHERE x IN (SELECT x FROM P WHERE y > 50);
```

Suppose there are 20 tuples where $y > 50$, and these 20 tuples have 10 distinct values of x. Among the 80 tuples where $y \leq 50$, exactly 15 tuples share those same 10 distinct values of x. How many tuples remain in P after deletion?
A. 65
B. 70
C. 80
D. 75

Problem 155 ***[MSQ]*** *Let $R(A, B)$ be a relation. Under which of the following conditions will the queries Q_1 and Q_2 always return the same result?*
Q_1: `SELECT DISTINCT A FROM R WHERE B = 'Active'`
Q_2: `SELECT A FROM R WHERE B = 'Active' GROUP BY A`
A. When A is a Primary Key of R.
B. When (A, B) is a Candidate Key of R.
C. Equal even when there are NULL values in R.
D. These two queries always return the same result for any instance of R.

Problem 156 ***[NAT]*** *Consider a table $T(a)$ with values $\{1, 2, 3, NULL\}$. Let Q be the query:*

```
SELECT COUNT(*) FROM T t1
WHERE NOT EXISTS (SELECT * FROM T t2 WHERE t2.a = t1.a);
```

The output is ____.

Problem 157 ***[MCQ]*** *Consider the schema $Account(branch, accNo, balance)$. The query:*

```
SELECT branch FROM Account a1
WHERE balance > ALL (SELECT balance FROM Account a2 WHERE a1.branch = a2.branch);
```

What does this query return?
A. Branches that have at least one account.
B. Branches where all balances are the same.
C. No rows (Empty set).
D. The branch with the absolute maximum balance in the entire table.

Problem 158 ***[NAT]*** *Consider the relation $R(A)$ with values $\{10, 20, 30, 40\}$.*

```
SELECT SUM(A) FROM R WHERE A < (SELECT MAX(A) FROM R);
```

If we add an additional tuple with value `NULL` to R, the new result of the query will be ____.

Problem 159 ***[MCQ]*** *Consider the table `Work(eid, hours)`.*

```
SELECT eid FROM Work
WHERE hours > (SELECT AVG(hours) FROM Work);
```

Assume the table has 100 rows and the `hours` values follow a strict arithmetic progression $1, 2, 3, \ldots, 100$. How many `eids` are returned?
A. 49
B. 50
C. 51
D. 100

Problem 160 ***[MSQ]*** *Which of the following statements about SQL constraints are TRUE?*
A. A `PRIMARY KEY` constraint implicitly includes a `UNIQUE` constraint and a `NOT NULL` constraint.
B. A table can have multiple `UNIQUE` constraints but only one `PRIMARY KEY`.
C. A `FOREIGN KEY` cannot reference a non-primary key column even if that column is marked `UNIQUE`.
D. `ON DELETE SET NULL` is only valid if the foreign key column allows NULL values.

Problem 161 ***[NAT]*** *Consider relation $R(A, B)$. There are 10 tuples with $B = 1$, 20 tuples with $B = 2$, and 30 tuples with $B = 3$. All A values are distinct.*

```
SELECT COUNT(*) FROM (
    SELECT B, AVG(A) as avgA FROM R GROUP BY B
    INTERSECT
    SELECT B, SUM(A)/COUNT(*) as avgA FROM R GROUP BY B
) AS temp;
```

The result is ____.

Problem 162 ***[MCQ]*** *Consider the relation $Suppliers(sid, sname, city)$.*

```
SELECT city, COUNT(*) FROM Suppliers
GROUP BY city
HAVING COUNT(*) > 1 OR city = 'Mumbai';
```

If the table has 3 suppliers from 'Mumbai', 1 from 'Delhi', and 1 from 'Bangalore', how many rows are in the output?
A. 1
B. 2
C. 3
D. 5

Problem 163 ***[NAT]*** *In SQL, the `LIKE` operator is used for pattern matching. Consider a table `Docs(title)` with titles: `'SQL_Standard'`, `'SQL100'`, `'SQL_2026'`, `'MySQL'`.*
How many rows are returned by:

```
SELECT * FROM Docs WHERE title LIKE 'SQL\_%';
```

Problem 164 ***[MCQ]*** *Consider the relation $S(a, b)$.*

```
SELECT a FROM S s1
WHERE b > (SELECT MIN(b) FROM S s2 WHERE s1.a = s2.a);
```

This query returns the 'a' values of all tuples in S except:
A. Those with the minimum 'b' value for each 'a'.
B. Those with the maximum 'b' value for each 'a'.
C. Those where 'a' appears only once in the table.
D. Both A and C.

Problem 165 ***[NAT]*** *Consider $T1(A, B)$ and $T2(C, D)$. $T1$ has 4 rows, $T2$ has 3 rows.*

```
SELECT COUNT(*) FROM T1 CROSS JOIN T2 WHERE T1.A = T2.C OR T1.A <> T2.C;
```

Assume there are no NULL values in column A and C. The result is ____.

Problem 166 ***[MSQ]*** *Which of the following conditions will result in the query `SELECT * FROM R, S` producing zero tuples?*
A. R is empty.
B. S is empty.
C. Both R and S are empty.
D. R and S have no common attributes.

Problem 167 ***[NAT]*** *Consider the table `Test(v)` with values $\{NULL, NULL, NULL\}$.*

```
SELECT COUNT(v) + COUNT(*) FROM Test;
```

The result is ____.

Problem 168 ***[MCQ]*** *Consider the following relational schema:*

```
Department (dept_id INT PRIMARY KEY, dept_name VARCHAR(20));
Employee (emp_id INT PRIMARY KEY, name VARCHAR(20), salary INT,
          dept_id INT, FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
          ON DELETE CASCADE);
```

If a department with `dept_id = 10` is deleted from the `Department` table, which of the following happens in the `Employee` table?
A. All rows with `dept_id = 10` are set to `NULL`.
B. All rows with `dept_id = 10` are deleted.
C. The deletion is prevented because it violates referential integrity.
D. Only the `dept_id` value in the Employee table is deleted, keeping the row.

Problem 169 ***[NAT]*** *Consider the table `T(A, B)` with the following instance:*

A	B
1	2
3	NULL
NULL	4
2	1

The number of rows returned by the following query is ____.

```
SELECT * FROM T WHERE A NOT IN (SELECT B FROM T);
```

Problem 170 ***[MSQ]*** *Consider the relation `Product(pid, pname, price, category)`. Which of the following SQL queries will return the `pname` of the most expensive product(s) in each category?*
A.

```
SELECT pname FROM Product P1 WHERE price =
(SELECT MAX(price) FROM Product P2 WHERE P1.category = P2.category)
```

B.

```
SELECT pname FROM Product WHERE price >= ALL
(SELECT price FROM Product GROUP BY category)
```

C.

```
SELECT P1.pname FROM Product P1,
(SELECT category, MAX(price) as maxp FROM Product GROUP BY category) P2
WHERE P1.category = P2.category AND P1.price = P2.maxp
```

D.

```
SELECT pname FROM Product WHERE price IN
(SELECT MAX(price) FROM Product GROUP BY category)
```

Problem 171 ***[NAT]*** *Consider a table `R(A)` with 10 distinct non-null integer values. A second table `S(B)` is created as:*

```
CREATE TABLE S AS SELECT A FROM R WHERE A > 100;
```

If the query `SELECT COUNT(*) FROM R WHERE NOT EXISTS (SELECT * FROM S WHERE S.B = R.A)` returns 7, how many tuples in `R` have a value of `A` greater than 100?

Problem 172 ***[MCQ]*** *Consider the table `Instructor(ID, name, dept_name, salary)`. The following query is executed:*

```
SELECT dept_name, AVG(salary)
FROM Instructor
GROUP BY dept_name
HAVING AVG(salary) > (SELECT AVG(salary) FROM Instructor);
```

Which of the following describes the output?
A. Departments where every instructor earns more than the company average.
B. Departments where the average salary is higher than the average salary of all instructors in the database.
C. Departments where the average salary is higher than the average salary of the lowest-paid department.
D. This query will result in a syntax error because `AVG` cannot be used in `HAVING`.

Problem 173 ***[NAT]*** *Consider the following tables:*

R

A
1
2
2

S

A
2
3

The number of tuples returned by `(SELECT A FROM R) EXCEPT ALL (SELECT A FROM S)` is ____.

Problem 174 ***[MCQ]*** *Consider the schema `Users(uid, age, city)`. The following update is attempted:*

```
UPDATE Users SET age = age + 1 WHERE city LIKE 'B%';
```

If the table initially contains `(1, 25, 'Bangalore')`, `(2, 30, 'Bombay')`, `(3, 22, 'Delhi')`, how many rows are modified?
A. 1
B. 2
C. 3
D. 0

Problem 175 ***[MCQ]*** *Given a relation `Emp(eid, name, sal, mgr_id)`, where `mgr_id` is a self-referencing foreign key to `eid`.*

```
SELECT E1.name
FROM Emp E1
WHERE NOT EXISTS (SELECT * FROM Emp E2 WHERE E2.mgr_id = E1.eid);
```

The query returns the names of:
A. Managers who manage no one.
B. Employees who are not managers.
C. The CEO (who has no manager).
D. All employees.

Problem 176 ***[MSQ]*** *Which of the following is true about the `GROUP BY` clause?*
A. Columns in the `SELECT` clause that are not part of an aggregate function must be present in the `GROUP BY` clause.
B. The `WHERE` clause filters rows before grouping, while `HAVING` filters groups after grouping.
C. `GROUP BY` automatically sorts the output by the grouping columns in ascending order in all SQL standards.
D. Aggregate functions can be used in the `WHERE` clause if a `GROUP BY` is present.

Problem 177 ***[MCQ]*** *Consider `R(A, B)` with 5 rows. `S(B, C)` has 10 rows. The size of `R NATURAL LEFT OUTER JOIN S` is at least:*
A. 0
B. 5
C. 10
D. 15

Problem 178 ***[NAT]*** *How many rows are returned by the following query if the table `Test(x)` contains values {`1, 2, 2, 3, NULL`}?*

```
SELECT x FROM Test WHERE x BETWEEN 1 AND 2;
```

Problem 179 ***[MCQ]*** *A table `Student(roll, name, score)` is updated using:*

```
ALTER TABLE Student ADD COLUMN grade CHAR(1) DEFAULT 'F';
```

If there were 100 students before this command, what is the value of `SELECT COUNT(grade) FROM Student`?
A. 0
B. 100
C. 1
D. NULL

Problem 180 ***[MSQ]*** *Consider `Sales(item, year, revenue)`. Which query finds items that had higher revenue in 2024 than in 2023?*
A.

```
SELECT S1.item FROM Sales S1, Sales S2
WHERE S1.item = S2.item AND S1.year = 2024
AND S2.year = 2023 AND S1.revenue > S2.revenue
```

B.

```
SELECT item FROM Sales WHERE year = 2024 AND revenue >
(SELECT revenue FROM Sales WHERE year = 2023)
```

C.

```
SELECT item FROM Sales S WHERE year = 2024 AND revenue >
(SELECT revenue FROM Sales WHERE year = 2023 AND item = S.item)
```

D.

```
SELECT item FROM Sales S WHERE year = 2024 AND revenue <
(SELECT revenue FROM Sales WHERE year = 2023 AND item = S.item)\\
```

Problem 181 ***[MCQ]*** *Consider the table `Staff(id, name, boss_id)`. The query:*

```
SELECT name FROM Staff S1
WHERE boss_id IS NULL AND EXISTS
(SELECT * FROM Staff S2 WHERE S2.boss_id = S1.id);
```

returns names of staff who:
A. Are bosses but have no boss themselves.
B. Have a boss and are also bosses.
C. Are at the bottom of the hierarchy.
D. Are the only employees in the company.

5.13 GATE PYQs

GATEPYQ 48 *Consider the following two relations, named* `Customer` *and* `Person`*, in a database:*

```
Person (
    aadhaar CHAR(12) PRIMARY KEY,
    name VARCHAR(32)
);

Customer (
    name VARCHAR(32),
    email VARCHAR(32) PRIMARY KEY,
    phone CHAR(10),
    aadhaar CHAR(12),
    FOREIGN KEY (aadhaar) REFERENCES Person(aadhaar)
);
```

Which of the following statements is/are correct? ***GATE DA 2025***

A. `aadhaar` *is a candidate key in the Customer relation*
B. `phone` *can be NULL in the Customer relation*
C. `aadhaar` *is a candidate key in the Person relation*
D. `aadhaar` *can be NULL in the Person relation*

GATEPYQ 49 *Consider a relation* `Loan` *of a bank:*

loan_number	branch_name	amount
L11	Banjara Hills	90000
L14	Kondapur	50000
L15	SR Nagar	40000
L22	SR Nagar	25000
L23	Balanagar	80000
L25	Kondapur	70000
L19	SR Nagar	65000

The following SQL query is executed:

```
SELECT L1.loan_number
FROM Loan L1
WHERE L1.amount > (
    SELECT MAX(L2.amount)
    FROM Loan L2
    WHERE L2.branch_name = 'SR Nagar'
);
```

The number of rows returned by the query is ____ (Answer in integer). ***GATE DA 2025***

GATEPYQ 50 *Consider the following relational schema:*

Students(rollno: integer, name: string, age: integer, cgpa: real)
Courses(courseno: integer, cname: string, credits: integer)
Enrolled(rollno: integer, courseno: integer, grade: string)

Which of the following options is/are correct SQL query/queries to retrieve the names of the students enrolled in course number (i.e. courseno) 1470? ***GATE CSE 2025***

*A. SELECT S.name FROM Students S WHERE EXISTS (SELECT * FROM Enrolled E WHERE E.courseno = 1470 AND E.rollno = S.rollno)*

*B. SELECT S.name FROM Students S WHERE SIZEOF (SELECT * FROM Enrolled E WHERE E.courseno = 1470 AND E.rollno = S.rollno) > 0*

C. SELECT S.name FROM Students S WHERE 0 < (SELECT COUNT(*) FROM Enrolled E WHERE E.courseno = 1470 AND E.rollno = S.rollno)

D. SELECT S.name FROM Students S NATURAL JOIN Enrolled E WHERE E.courseno = 1470

GATEPYQ 51 *Consider the following database tables of a sports league.*
player(pid, pname, age) *team(tid, tname, city, cid)*
coach(cid, cname) *members(pid, tid)*
An instance of the table and an SQL query are given.

player

pid	pname	age
1	*Jasprit*	31
2	*Atharva*	24
3	*Ishan*	26
4	*Axar*	30

coach

cid	cname
101	*Ricky*
102	*Mark*
103	*Trevor*

team

tid	tname	city	cid
10	*MI*	*Mumbai*	102
20	*DC*	*Delhi*	101
30	*PK*	*Mohali*	103

members

pid	tid
1	10
2	30
3	10
4	20

```
SELECT MIN (P.age)
FROM player P
WHERE P.pid IN (
    SELECT M.pid
    FROM team T, coach C, members M
    WHERE C.cname = 'Mark'
          AND T.cid = C.cid
          AND M.tid = T.tid
    )
```

The value returned by the given SQL query is _____. ***GATE CSE 2025***

GATEPYQ 52 *Consider the following two tables named `Raider` and `Team` in a relational database maintained by a Kabaddi league. The attribute `ID` in table `Team` references the primary key of the `Raider` table, `ID`.*

Raider

ID	Name	Raids	RaidPoints
1	Arjun	200	250
2	Ankush	190	219
3	Sunil	150	200
4	Reza	150	190
5	Pratham	175	220
6	Gopal	193	215

Team

City	ID	BidPoints
Jaipur	2	200
Patna	3	195
Hyderabad	5	175
Jaipur	1	250
Patna	4	200
Jaipur	6	200

The following SQL query is executed:

```
SELECT *
FROM Raider, Team
WHERE Raider.ID = Team.ID
  AND City = 'Jaipur'
  AND RaidPoints > 200;
```

The number of rows returned by this query is ____ . ***GATE DA 2024***

GATEPYQ 53 *Consider the following table named Student in a relational database. The primary key of this table is rollNum.*

rollNum	name	gender	marks
1	Naman	M	62
2	Aliya	F	70
3	Aliya	F	80
4	James	M	82
5	Swati	F	65

The SQL query below is executed on this database.

```
SELECT *
FROM Student
WHERE gender = 'F' AND
      marks > 65;
```

The number of rows returned by the query is _____. ***GATE CSE 2023***

GATEPYQ 54 *Consider the relational database with the following four schemas and their respective instances.*

Students		
sNo	**sName**	**dNo**
$S01$	$James$	$D01$
$S02$	$Rocky$	$D01$
$S03$	$Jackson$	$D02$
$S04$	$Jane$	$D01$
$S05$	$Milli$	$D02$

Dept	
dNo	**dName**
$D01$	CSE
$D02$	EEE

Course		
cNo	**cName**	**dNo**
$C11$	DS	$D01$
$C12$	OS	$D01$
$C21$	DE	$D02$
$C22$	PT	$D02$
$C23$	CV	$D03$

Register	
sNo	**cNo**
$S01$	$C11$
$S01$	$C12$
$S02$	$C11$
$S03$	$C21$
$S03$	$C22$
$S03$	$C23$
$S04$	$C11$
$S04$	$C12$
$S05$	$C11$
$S05$	$C21$

Student(sNo, sName, dNo) *Dept(dNo, dName)*
Course(cNo, cName, dNo) *Register(sNo, cNo)*

```
SELECT *
FROM Student AS S
WHERE NOT EXISTS
(
    SELECT cNo FROM Course WHERE dNo = "D01"
    EXCEPT
    SELECT cNo FROM Register WHERE sNo = S.sNo
)
```

The number of rows returned by the above query is _____. ***GATE CSE 2022***

GATEPYQ 55 *The relation scheme given below is used to store information about the employees of a company, where empId is the key and deptId indicates the department to which the employee is assigned. Each employee is assigned to exactly one department.*

emp(empId, name, gender, salary, deptId)

Consider the following SQL query:

```
select deptId, count(*)
from emp
where gender = "female"
  and salary > (select avg(salary) from emp)
group by deptId;
```

The above query gives, for each department in the company, the number of female employees whose salary is greater than the average salary of ***GATE CSE 2021***

A. employees in the department
B. employees in the company
C. female employees in the department
D. female employees in the company

GATEPYQ 56 *Consider a relational database containing the following schemas. The primary key of each table is indicated by underlining the constituent fields.*

Catalogue

sno	pno	cost
$S1$	$P1$	150
$S1$	$P2$	50
$S1$	$P3$	100
$S2$	$P4$	200
$S2$	$P5$	250
$S3$	$P1$	250
$S3$	$P2$	150
$S3$	$P5$	300
$S3$	$P4$	250

Suppliers

sno	sname	location
$S1$	*M/s Royal furniture*	$Delhi$
$S2$	*M/s Balaji furniture*	$Bangalore$
$S3$	*M/s Premium furniture*	$Chennai$

Parts

pno	pname	part_spec
$P1$	$Table$	$Wood$
$P2$	$Chair$	$Wood$
$P3$	$Table$	$Steel$
$P4$	$Almirah$	$Steel$
$P5$	$Almirah$	$Wood$

```
SELECT s.sno, s.sname
FROM Suppliers s, Catalogue c
WHERE s.sno = c.sno
  AND cost > (SELECT AVG(cost)
              FROM Catalogue
              WHERE pno = 'P4'
              GROUP BY pno);
```

The number of rows returned by the above SQL query is ***GATE CSE 2020***

A. 4
B. 5
C. 0
D. 2

GATEPYQ 57 *Consider the following two tables and four queries in SQL.*
Book(isbn, bname), Stock(isbn, copies)

Query 1:

```
SELECT B.isbn, S.copies
FROM Book B INNER JOIN Stock S
ON B.isbn = S.isbn;
```

Query 2:

```
SELECT B.isbn, S.copies
FROM Book B LEFT OUTER JOIN Stock S
ON B.isbn = S.isbn;
```

Query 3:

```
SELECT B.isbn, S.copies
FROM Book B RIGHT OUTER JOIN Stock S
ON B.isbn = S.isbn;
```

Query 4:

```
SELECT B.isbn, S.copies
FROM Book B FULL OUTER JOIN Stock S
ON B.isbn = S.isbn;
```

Which one of the queries above is certain to have an output that is a superset of the outputs of the other three queries?

GATE CSE 2018

(A) Query 1

(B) Query 2

(C) Query 3

(D) Query 4

GATEPYQ 58 *Consider the following database table named top_scorer.*

top_scorer

player	***country***	***goals***
Klose	*Germany*	16
Ronaldo	*Brazil*	15
G Muller	*Germany*	14
Fontaine	*France*	13
Pele	*Brazil*	12
Klinsmann	*Germany*	11
Kocsis	*Hungary*	11
Batistuta	*Argentina*	10
Cubillas	*Peru*	10
Lato	*Poland*	10
Lineker	*England*	10
T Muller	*Germany*	10
Rahn	*Germany*	10

Consider the following SQL query:

```
SELECT ta.player FROM top_scorer AS ta
WHERE ta.goals > ALL (
    SELECT tb.goals
    FROM top_scorer AS tb
    WHERE tb.country = 'Spain'
)
AND ta.goals > ANY (
    SELECT tc.goals
    FROM top_scorer AS tc
    WHERE tc.country = 'Germany'
);
```

The number of tuples returned by the above SQL query is _____. (GATE CSE 2017)

GATEPYQ 59 *Consider a database that has the relation schema EMP(EmpId, EmpName, DeptName). An instance of the schema EMP and a SQL query on it are given below.*

EMP

EmpId	*EmpName*	*DeptName*
1	XYA	AA
2	XYB	AA
3	XYC	AA
4	XYD	AA
5	XYE	AB
6	XYF	AB
7	XYG	AB
8	XYH	AC
9	XYI	AC
10	XYJ	AC
11	XYK	AD
12	XYL	AD
13	XYM	AE

```
SELECT AVG(EC.Num)
FROM EC
WHERE (DeptName, Num) IN
    (SELECT DeptName, COUNT(EmpId) AS Num
     FROM EMP
     GROUP BY DeptName) AS EC(DeptName, Num);
```

The output of executing the SQL query is _____. (GATE CSE 2017)

GATEPYQ 60 *Consider the following database table named water_schemes.*

Water_schemes

scheme_no	*district_name*	*capacity*
1	*Ajmer*	20
1	*Bikaner*	10
2	*Bikaner*	10
3	*Bikaner*	20
1	*Churu*	10
2	*Churu*	20
1	*Dungargarh*	10

The number of tuples returned by the following SQL query is _____. (GATE CSE 2016)

```
WITH total(name, capacity) AS
     (SELECT district_name, SUM(capacity)
      FROM water_schemes
      GROUP BY district_name),
total_avg(capacity) AS
     (SELECT AVG(capacity)
      FROM total)
SELECT name
FROM total, total_avg
WHERE total.capacity >= total_avg.capacity;
```

GATEPYQ 61 *Consider the following relation*
Cinema(theater, address, capacity)
Which of the following options will be needed at the end of the SQL query

```
SELECT P1.address
FROM Cinema P1
```

such that it always finds the addresses of theaters with maximum capacity?

(A) WHERE P1.capacity >= ALL (SELECT P2.capacity FROM Cinema P2)

(B) WHERE P1.capacity >= ANY (SELECT P2.capacity FROM Cinema P2)

(C) WHERE P1.capacity > ALL (SELECT MAX(P2.capacity) FROM Cinema P2)

(D) WHERE P1.capacity > ANY (SELECT MAX(P2.capacity) FROM Cinema P2)

GATE CSE 2015

GATEPYQ 62 *Consider the following relation:*

Student(Roll_No, Student_Name, . . .) *Performance(Roll_No, Subject_Code, Marks)*

Student

Roll_No	Student_Name
1	*Raj*
2	*Rohit*
3	*Raj*

Performance

Roll_No	Course	Marks
1	*Math*	80
1	*English*	70
2	*Math*	75
3	*English*	80
2	*Physics*	65
3	*Math*	80

Consider the following SQL query.

```
SELECT S.Student_Name, SUM(P.Marks)
FROM Student S, Performance P
WHERE S.Roll_No = P.Roll_No
GROUP BY S.Student_Name;
```

The number of rows that will be returned by the SQL query is

GATECSE2015

GATEPYQ 63 *SELECT operation in SQL is equivalent to*

A. *the selection operation in relational algebra*

B. *the selection operation in relational algebra, except that SELECT in SQL retains duplicates*

C. *the projection operation in relational algebra*

D. *the projection operation in relational algebra, except that SELECT in SQL retains duplicates*

GATE CSE 2015

GATEPYQ 64 *Consider the following relational schema:*
Employee(empId, empName, empDept) Customer(custId, custName, salesRepId, rating)
SalesRepId is a foreign key referring to empId of the Employee relation. Assume that each employee makes a sale to at least one customer. What does the following query return? GATE CSE 2014

```
SELECT empName
FROM employee E
WHERE NOT EXISTS (
    SELECT custId
    FROM customer C
    WHERE C.salesRepId = E.empId
      AND C.rating <> 'GOOD'
);
```

A. *Names of all the employees with at least one of their customers having a 'GOOD' rating*

B. *Names of all the employees with at most one of their customers having a 'GOOD' rating*

C. *Names of all the employees with none of their customers having a 'GOOD' rating*

D. *Names of all the employees with all their customers having a 'GOOD' rating*

GATEPYQ 65 *SQL allows duplicate tuples in relations, and correspondingly defines the multiplicity of tuples in the result of joins. Which one of the following queries always gives the same answer as the nested query shown below? GATE CSE 2014*

```
SELECT *
FROM R
WHERE a IN (SELECT S.a FROM S);
```

A. *SELECT R.* FROM R, S WHERE R.a = S.a*

B. *SELECT DISTINCT R.* FROM R, S WHERE R.a = S.a*

C. *SELECT R.* FROM R, (SELECT DISTINCT a FROM S) AS S1 WHERE R.a = S1.a*

D. *SELECT R.* FROM R, S WHERE R.a = S.a AND IS UNIQUE R*

GATEPYQ 66 *Given the following schema: GATE CSE 2014*
employees(emp-id, first-name, last-name, hire-date, dept-id, salary)
departments(dept-id, dept-name, manager-id, location-id)
You want to display the last names and hire dates of all latest hires in their respective departments in the location ID 1700. You issue the following query:

```
SELECT last-name, hire-date
FROM employees
WHERE (dept-id, hire-date) IN
(
  SELECT dept-id, MAX(hire-date)
  FROM employees JOIN departments USING(dept-id)
  WHERE location-id = 1700
  GROUP BY dept-id
);
```

What is the outcome?

A. *It executes but does not give the correct result*

B. *It executes and gives the correct result*

C. *It generates an error because of pairwise comparison*

D. *It generates an error because the GROUP BY clause cannot be used with table joins in a sub-query*

GATEPYQ 67 *Given the following statements: GATE CSE 2014*
S1: A foreign key declaration can always be replaced by an equivalent check assertion in SQL. S2: Given the table R(a,b,c) where a and b together form the primary key, the following is a valid table definition.

```
CREATE TABLE S (
a INTEGER,
d INTEGER,
e INTEGER,
PRIMARY KEY (d),
FOREIGN KEY (a) REFERENCES R
);
```

Which one of the following statements is CORRECT?

A. *S1 is TRUE and S2 is FALSE*

B. *Both S1 and S2 are TRUE*

C. *S1 is FALSE and S2 is TRUE*

D. Both S1 and S2 are FALSE

GATEPYQ 68 *Consider the following relations A, B and C.*
How many tuples does the result of the following SQL query contain? ***GATE CSE 2012***

A

Id	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

B

Id	Name	Age
15	Shreya	24
25	Hari	40
98	Rohit	20
99	Rohit	11

C

Id	Phone	Area
10	2200	02
99	2100	01

```
SELECT A.Id
FROM A
WHERE A.Age > ALL (
    SELECT B.Age
    FROM B
    WHERE B.Name = 'Arun'
);
```

A. 4

B. 3

C. 0

D. 1

GATEPYQ 69 *Which of the following statements are TRUE about an SQL query? GATE CSE 2012*
P : An SQL query can contain a HAVING clause even if it does not have a GROUP BY clause Q : An SQL query can contain a HAVING clause only if it has a GROUP BY clause R : All attributes used in the GROUP BY clause must appear in the SELECT clause S : Not all attributes used in the GROUP BY clause need to appear in the SELECT clause

A. P and R

B. P and S

C. Q and R

D. Q and S

GATEPYQ 70 *Database table by name Loan_Records is given below. What is the output of the following SQL query?* ***GATE CSE 2011***

Borrower	Bank_Manager	Loan_Amount
Ramesh	Sunderajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00

```
SELECT count(*)
FROM(
 (SELECT Borrower, Bank_Manager FROM Loan_Records) AS S
 NATURAL JOIN
 (SELECT Bank_Manager, Loan_Amount FROM Loan_Records) AS T
);
```

A. 3

B. 9

C. 5

D. 6

GATEPYQ 71 *Consider a database table T containing two columns X and Y each of type integer. After the creation of the table, one record (X=1,Y=1) is inserted in the table. Let MX and MY denote the respective maximum values of X and Y among all records in the table at any point in time. Using MX and MY, new records are inserted in the table 128 times with X and Y values being MX+1, 2*MY+1 respectively. It may be noted that each time after the insertion, values of MX and MY change. What will be the output of the following SQL query after the steps mentioned above are carried out?* ***GATE CSE 2011***

```
SELECT Y FROM T WHERE X=7;
```

A. 127

B. 255

C. 129

D. 257

GATEPYQ 72 *A relational schema for a train reservation database is given below. What pids are returned by the following SQL query for the given instance of the tables?* ***GATE CSE 2010***

Passenger(*pid*, *pname*, *age*)

Reservation(*pid*, *class*, *tid*)

Table : Passenger

pid	*pname*	*Age*
0	'Sachin'	65
1	'Rahul'	66
2	'Sourav'	67
3	'Anil'	69

Table : Reservation

pid	*class*	*tid*
0	'AC'	8200
1	'AC'	8201
2	'SC'	8201
5	'AC'	8203
1	'SC'	8204
3	'AC'	8202

```
SELECT pid
FROM Reservation
WHERE class = 'AC' AND
      EXISTS (
          SELECT *
          FROM Passenger
          WHERE age > 65
            AND Passenger.pid = Reservation.pid
      );
```

A. 1,0

B. 1,2

C. 1,3

D. 1,5

GATEPYQ 73 *Consider the following relational schema and SQL query. Which one of the following is the correct interpretation of the query? GATE CSE 2009*

Suppliers(sid:integer, sname:string, city:string, street:string)
Parts(pid:integer, pname:string, color:string)
Catalog(sid:integer, pid:integer, cost:real)

```
SELECT S.sname
FROM Suppliers S
WHERE S.sid NOT IN (
    SELECT C.sid
    FROM Catalog C
    WHERE C.pid NOT IN (
        SELECT P.pid
        FROM Parts P
        WHERE P.color <> 'blue'
    )
);
```

A. *Find the names of all suppliers who have supplied a non-blue part.*

B. *Find the names of all suppliers who have not supplied a non-blue part.*

C. *Find the names of all suppliers who have supplied only blue parts.*

D. *Find the names of all suppliers who have not supplied only blue parts.*

GATEPYQ 74 *Consider the table employee(empId, name, department, salary) and the two queries Q1 and Q2 below. Assuming that department 5 has more than one employee, and we want to find the employees who get higher salary than anyone in department 5, which one of the following statements is TRUE for any arbitrary employee table? GATE CSE 2007*

Q1

```
Select e.empId
From employee e
Where not exists
 (Select * From employee s
  where s.department = '5'
    and s.salary >= e.salary);
```

Q2

```
Select e.empId
From employee e
Where e.salary > Any
 (Select distinct salary
  From employee s
  Where s.department = '5');
```

A. *Q1 is the correct query*

B. *Q2 is the correct query*

C. *Both Q1 and Q2 produce the same answer*

D. *Neither Q1 nor Q2 is the correct query*

GATEPYQ 75 *Consider the relation enrolled(student, course) in which (student, course) is the primary key, and the relation paid(student, amount) where student is the primary key. Assume no NULL values and no foreign keys or integrity constraints. Given the following four queries:* ***GATE CSE 2006***

Query 1

```
Select * from enrolled
where student in (select student from paid);
```

Query 2

```
Select student from paid
where student in (select student from enrolled);
```

Query 3

```
Select E.student
from enrolled E, paid P
where E.student = P.student;
```

Query 4

```
Select student from paid
where exists
 (select * from enrolled
  where enrolled.student = paid.student);
```

A. *All queries return identical row sets for any database*

B. *Query 2 and Query 4 return identical row sets for all databases but there exist databases for which Query 1 and Query 2 return different row sets*

C. *There exist databases for which Query 3 returns strictly fewer rows than Query 2*

D. *There exist databases for which Query 4 will encounter an integrity violation at runtime*

GATEPYQ 76 *The relation book(title, price) contains the titles and prices of different books. Assuming that no two books have the same price, what does the following SQL query list?* ***GATE CSE 2005***

```
select title
from book as B
where (select count(*)
       from book as T
       where T.price > B.price) < 5;
```

A. Titles of the four most expensive books
B. Title of the fifth most inexpensive book
C. Title of the fifth most expensive bookTitles of the five most expensive books
D. Titles of the five most expensive books

GATEPYQ 77 *Consider the following relation schema pertaining to a students database:*

$$\textit{Students}(\underline{rollno}, \textit{name}, \textit{address})$$

$$\textit{Enroll}(\underline{rollno}, \underline{courseno}, \textit{coursename})$$

Where the primary keys are shown underlined. The number of tuples in the Students and Enroll tables are 120 and 8 respectively. What are the maximum and minimum number of tuples that can be present in (Students $*$ Enroll), where '' denotes natural join?* ***GATE CSE 2004***
A. 8, 8
B. 120, 8
C. 960, 8
D. 960, 120

GATEPYQ 78 *Consider the set of relations shown below and the SQL query that follows:*

$$\textit{Students}(\underline{Roll\_number}, \textit{Name}, \textit{Date\_of\_birth})$$

$$\textit{Courses}(\underline{Course\_number}, \textit{Course\_name}, \textit{Instructor})$$

$$\textit{Grades}(\underline{Roll\_number, Course\_number}, \textit{Grade})$$

```
select distinct Name
from Students, Courses, Grades
where Students.Roll_number = Grades.Roll_number
  and Courses.Instructor = 'Korth'
  and Courses.Course_number = Grades.Course_number
  and Grades.grade = 'A';
```

Which of the following sets is computed by the above query? ***GATE CSE 2003***
A. Names of students who have got an A grade in all courses taught by Korth
B. Names of students who have got an A grade in all courses
C. Names of students who have got an A grade in at least one of the courses taught by Korth
D. None of the above

GATEPYQ 79 *Given relations $r(w, x)$ and $s(y, z)$, consider the SQL query:*

```
select distinct w, x from r, s;
```

The result of the above query is guaranteed to be the same as r, provided: ***GATE CSE 2000***
A. r has no duplicates and s is non-empty
B. r and s have no duplicates
C. s has no duplicates and r is non-empty
D. r and s have the same number of tuples

5.14 Try it Yourself

Exercise 138 ***[NAT]*** *Consider a table $R(A)$ with values $\{1, 2, 2, 3, NULL, NULL\}$.*
What is the result of the query: `SELECT COUNT(A) + COUNT(*) FROM R;`

Exercise 139 ***[MCQ]*** *Consider the relations $R(x, y)$ and $S(y, z)$. The query:*
`SELECT x FROM R WHERE y NOT IN (SELECT y FROM S);`
If the subquery `(SELECT y FROM S)` *returns at least one* `NULL` *value, how many rows will the outer query return?*
A. All rows of R
B. Only rows of R where y is NULL
C. Zero rows
D. Rows of R where y does not match any non-null value in S

Exercise 140 ***[NAT]*** *Let $R(A, B)$ be a relation with 20 tuples. If we execute:*
`SELECT DISTINCT * FROM R;`
and it returns 20 tuples, what is the maximum number of tuples returned by:
`SELECT A, B FROM R GROUP BY A, B;`

Exercise 141 ***[MSQ]*** *Which of the following statements regarding the* `HAVING` *clause are correct?*
A. It can be used without a `GROUP BY` *clause.*
B. It filters rows before the aggregate functions are calculated.
C. It can contain aggregate functions like `SUM` *or* `AVG`*.*
D. Any column appearing in the `HAVING` *clause must also appear in the* `GROUP BY` *clause (if not aggregated).*

Exercise 142 ***[NAT]*** *Consider the tables $T1$ and $T2$ both having a single column A.*
$T1 = \{1, 2, 3\}$, $T2 = \{2, 3, 4\}$.
How many rows are returned by: `(SELECT A FROM T1) UNION ALL (SELECT A FROM T2);`

Exercise 143 ***[MCQ]*** *A table* `Employee(id, salary)` *has 10 rows with distinct salaries. The query:*

```
SELECT id FROM Employee E1
WHERE 2nd = (SELECT COUNT(DISTINCT salary) FROM Employee E2 WHERE E2.salary >=
E1.salary);
```

(Note: Assume '2nd' is the literal value 2). What does this query find?
A. The employee with the minimum salary.
B. The employee with the 2nd highest salary.
C. The employee with the 2nd lowest salary.
D. No rows due to syntax error.

Exercise 144 ***[NAT]*** *Consider $R(A)$ with values $\{10, 20, 30\}$. What is the output of:*
`SELECT AVG(A) FROM R WHERE A > 30;`
(Note: If the result is NULL, enter 0).

Exercise 145 ***[MSQ]*** *Which of the following SQL snippets will successfully delete all rows from a table named* `Logs`*?*
A. `DELETE FROM Logs;`
B. `DROP TABLE Logs;`
C. `TRUNCATE TABLE Logs;`
D. `REMOVE * FROM Logs;`

Exercise 146 ***[NAT]*** *Consider two relations $R(A, B)$ and $S(C, D)$. R has 5 rows and S has 3 rows.*
What is the number of rows in the result of `SELECT * FROM R, S WHERE 1=0;`

Exercise 147 ***[MCQ]*** *In a `NATURAL JOIN` between $R(A, B, C)$ and $S(C, D, E)$, the joining attribute is:*
A. A
B. B
C. C
D. No attribute (it results in a Cartesian product)

Exercise 148 ***[NAT]*** *Consider a table `Sales(qty)` with values $\{10, 20, NULL\}$.*
What is the value of `SELECT SUM(qty) / COUNT(*) FROM Sales;`

Exercise 149 ***[MCQ]*** *Which of the following is equivalent to the `INTERSECT` operator for relations $R(A)$ and $S(A)$?*
A. `SELECT A FROM R WHERE A IN (SELECT A FROM S)`
B. `SELECT A FROM R WHERE A NOT IN (SELECT A FROM S)`
C. `SELECT A FROM R UNION SELECT A FROM S`
D. `SELECT A FROM R, S`

Exercise 150 ***[NAT]*** *A table `Data(x)` has 100 rows. A query `SELECT COUNT(DISTINCT x) FROM Data` returns 20. If we update 5 rows that had distinct values to a value already present in the remaining 15 distinct values, what will the new `COUNT(DISTINCT x)` return?*

Exercise 151 ***[MSQ]*** *Consider the `LIKE` operator in SQL. Which of the following patterns will match the string `'DB_2026'`?*
A. `'DB_202%'`
B. `'DB__2026'`
C. `'D%26'`
D. `'DB_202_'`

Exercise 152 ***[NAT]*** *Consider the table `Student(name, marks)`. There are 5 students with marks $\{70, 80, 80, 90, 100\}$. How many rows are returned by: `SELECT marks FROM Student GROUP BY marks HAVING marks > 75;`*

Exercise 153 ***[MCQ]*** *What is the default ordering of the `ORDER BY` clause?*
A. Randomized
B. Descending (DESC)
C. Ascending (ASC)
D. Based on Primary Key

Exercise 154 ***[NAT]*** *In the table `T1(val)` with values $\{1, 2, 3\}$, how many rows are returned by:*
`SELECT * FROM T1 AS A, T1 AS B WHERE A.val < B.val;`

Exercise 155 ***[MCQ]*** *Consider `R(A)` and `S(A)`. The operation `R EXCEPT S` is equivalent to:*
A. $R - (R \cap S)$
B. $S - R$
C. $R \cup S$
D. $R \cap S$

Exercise 156 ***[NAT]*** *Consider a table `Votes(choice)` with values $\{$`'Yes', 'No', 'Yes', NULL, 'No'`$\}$.*
What is the result of `SELECT COUNT(DISTINCT choice) FROM Votes;`

Exercise 157 *Consider the table `Employee(empId, name, deptId, salary)`:*

empId	name	deptId	salary
1	Alice	10	75000
2	Bob	20	60000
3	Charlie	10	80000
4	David	20	55000
5	Eve	30	90000

Query:

```
SELECT COUNT(*)
FROM Employee
WHERE salary > 70000;
```

Find the numeric answer.
Answer: ____ *(NAT)*

Exercise 158 *Consider the table* `Sales(saleId, custId, productId)`*:*

saleId	custId	productId
1	101	10
2	102	10
3	101	11
4	103	12
5	104	10
6	101	12

Query:

```
SELECT COUNT(DISTINCT custId)
FROM Sales
WHERE productId = 10;
```

Answer: ____ *(NAT)*

Exercise 159 *Tables:* `Employee(empId, name, deptId, salary)`, `Department(deptId, deptName, budget)`

deptId	deptName	budget
10	HR	200000
20	IT	300000
30	Sales	250000

Query:

```
SELECT COUNT(*)
FROM Employee e JOIN Department d
ON e.deptId = d.deptId
WHERE salary > 70000 AND budget > 250000;
```

Answer: ____ *(NAT)*

Exercise 160 *Employee(empId, name, deptId, salary)*

empId	name	deptId	salary
1	Alice	10	75000
2	Bob	10	80000
3	Charlie	20	60000
4	David	20	55000
5	Eve	30	90000

Query:

```
SELECT deptId
FROM Employee
GROUP BY deptId
HAVING AVG(salary) > 70000;
```

Answer: ____ *(NAT)*

Exercise 161 *Employee(empId, name, managerId, salary)*

empId	name	managerId	salary
1	Alice	2	90000
2	Bob	NULL	80000
3	Charlie	2	75000
4	David	2	85000
5	Eve	4	90000

Query:

```
SELECT COUNT(*)
FROM Employee e1, Employee e2
WHERE e1.managerId = e2.empId AND e1.salary > e2.salary;
```

Answer: ____ *(NAT)*

Exercise 162 *Employee(empId, deptId, salary)*

empId	deptId	salary
1	10	75000
2	10	80000
3	20	60000
4	20	55000
5	30	90000
6	30	85000

Query:

```
SELECT COUNT(*)
FROM Employee
WHERE deptId IN (
  SELECT deptId
  FROM Employee
  GROUP BY deptId
  HAVING AVG(salary) > 80000
);
```

Answer: ____ *(NAT)*

Exercise 163 *Orders(orderId, custId, amount)*

orderId	custId	amount
101	1	5000
102	2	7000
103	1	9000
104	3	4000
105	2	6000
106	3	8000

Query:

```
SELECT COUNT(*)
FROM Orders
WHERE custId NOT IN (
  SELECT custId
  FROM Orders
  WHERE amount > 8000
);
```

Answer: ____ *(NAT)*

Exercise 164 *Student(rollNo, name), Grades(rollNo, courseId, grade)*

rollNo	name
1	Alice
2	Bob
3	Charlie

rollNo	courseId	grade
1	C1	A
1	C2	B
2	C1	B
2	C2	C
3	C1	A
3	C2	A

Query:

```
SELECT COUNT(*)
FROM Student S
WHERE EXISTS (
  SELECT *
  FROM Grades G
  WHERE G.rollNo = S.rollNo AND G.grade = 'A'
);
```

Answer: ____ *(NAT)*

Exercise 165 *Query:*

```
SELECT COUNT(*)
FROM Student S
WHERE NOT EXISTS (
  SELECT *
  FROM Grades G
  WHERE G.rollNo = S.rollNo AND G.grade = 'C'
);
```

Answer: ____ *(NAT)*

Exercise 166 *Product(productId, price)*

productId	*price*
1	*500*
2	*700*
3	*600*
4	*800*
5	*900*

Query:

```
SELECT COUNT(*)
FROM Product
WHERE price > ANY (
  SELECT price
  FROM Product
  WHERE productId IN (1,2)
);
```

Answer: ____ *(NAT)*

Exercise 167 *Query:*

```
SELECT COUNT(*)
FROM Product
WHERE price > ALL (
  SELECT price
  FROM Product
  WHERE productId IN (1,2)
);
```

Answer: ____ *(NAT)*

Exercise 168

```
SELECT COUNT(*)
FROM Product
WHERE price >= SOME (
  SELECT price
  FROM Product
  WHERE productId IN (2,3)
);
```

Answer: ____ *(NAT)*

Exercise 169 *Sales(saleId, empId, amount)*

saleId	empId	amount
1	1	5000
2	2	7000
3	1	9000
4	2	4000
5	3	8000
6	3	8500

Query:

```
SELECT COUNT(*)
FROM Sales S1
WHERE amount = (
  SELECT MAX(amount)
  FROM Sales S2
  WHERE S1.empId = S2.empId
);
```

Answer: ____ *(NAT)*

Exercise 170 *DeptA(empId), DeptB(empId)*

DeptA: 1,2,3,4
DeptB: 3,4,5,6

Query:

```
SELECT COUNT(*)
FROM (
  SELECT empId FROM DeptA
  UNION
  SELECT empId FROM DeptB
) AS AllEmp;
```

Answer: ____ *(NAT)*

Exercise 171 *Query:*

```
SELECT COUNT(*)
FROM (
  SELECT empId FROM DeptA
  INTERSECT
  SELECT empId FROM DeptB
) AS CommonEmp;
```

Answer: ____ *(NAT)*

Exercise 172 *Employee(empId, managerId)*

empId	managerId
1	2
2	NULL
3	2
4	3
5	NULL

Query:

```
SELECT COUNT(*)
FROM Employee
WHERE managerId IS NULL;
```

Answer: ____ *(NAT)*

Exercise 173 *Student(rollNo, deptId)*

rollNo	deptId
1	10
2	20
3	10
4	20
5	10

Query:

```
SELECT COUNT(*)
FROM (
  SELECT deptId, COUNT(*) AS Num
  FROM Student
  GROUP BY deptId
  HAVING COUNT(*) > 2
) AS T;
```

Answer: ____ *(NAT)*

Exercise 174 *Orders(orderId, custId, amount)*

orderId	custId	amount
101	1	5000
102	2	7000
103	1	9000
104	3	4000
105	2	6000
106	3	8000

Query:

```
SELECT COUNT(*)
FROM Orders
WHERE custId NOT IN (
  SELECT custId
  FROM Orders
  GROUP BY custId
  HAVING MAX(amount) > 8000
);
```

Answer: ____ *(NAT)*

Exercise 175 *Employee(empId, deptId, salary)*, *Department(deptId, budget)*

empId	deptId	salary
1	10	75000
2	10	80000
3	20	60000
4	20	55000
5	30	90000

Department(deptId, budget)

deptId	budget
10	200000
20	150000
30	300000

Query:

```
SELECT COUNT(*)
FROM Employee e
WHERE EXISTS (
```

```
  SELECT *
  FROM Department d
  WHERE e.deptId = d.deptId AND d.budget > 250000
);
```

Answer: ____ (NAT)

Exercise 176 *Sales(saleId, empId, amount)*

saleId	empId	amount
1	1	5000
2	1	7000
3	2	6000
4	2	8000
5	3	9000
6	3	8500

Query:

```
SELECT COUNT(*)
FROM Sales S1
WHERE amount = (
  SELECT MAX(amount)
  FROM Sales S2
  WHERE S1.empId = S2.empId
);
```

Answer: ____ (NAT)

5.15 YouTube Links and QR Codes

Lecture	Details	YouTube Link	QR Code
28	SQL Basics — CREATE, INSERT, DELETE, UPDATE, ALTER, & Aggregate Functions	https://youtu.be/jL1-gBaFsGY	
29	GROUP BY & HAVING Clause — Order of Execution in SQL	https://youtu.be/qHxj17q_9ss	
30	NULL Values in SQL — IS NULL, IS NOT NULL, NULL Handling	https://youtu.be/wJyaO7eOgWI	
31	Cartesian Product & Joins in SQL — Natural, Inner, Outer Joins	https://youtu.be/HP-rvF_7bQc	

32	SET Operations in SQL — UNION, UNION ALL, INTERSECT, EXCEPT	https://youtu.be/qAKE7yI2zZU	
33	Subqueries in SQL — Correlated & Non-Correlated — IN, ANY, ALL, EXISTS	https://youtu.be/fb92CQhfFH0	
34	Practice Problems on SQL	https://youtu.be/SBm67noqA6A	
35	GATE PYQs on SQL	https://youtu.be/qFpTahjWSHc	

Chapter 6

Normalization

Normalization is a systematic method for designing a relational database. Its main goal is to:

- Remove redundancy
- Avoid anomalies in insertion, update, and deletion
- Ensure data integrity

Advantages of Data Normalization

- Improved overall database organization: After normalization, your database will be structured and arranged in a way that is logical for all departments company-wide. With increased organization, duplication and location errors will be minimized, and outdated versions of data can be more easily updated.
- Data consistency: Consistent data is crucial for all teams within a business to stay on the same page. Data normalization will ensure consistency across development, research, and sales teams. Consistent data will also improve workflow between departments and align their information sets.
- Reduces redundancy: Redundancy is a commonly overlooked data storage issue. Reducing redundancy will ultimately help reduce file size and therefore speed up analysis and data processing time.
- Cost reduction: Cost reduction due to normalization involves a culmination of the previously mentioned benefits. For instance, if file size is reduced, data storage and processors won't need to be as large. Additionally, increased workflow due to consistency and organization will ensure that all employees are able to access the database information as quickly as possible, saving time for other necessary tasks.
- Increased security: Because normalization requires that data is more accurately located and uniformly organized, security is significantly increased.

Approach to Normalization

1. Decide if a given relation schema is in **good form** (Normal Form)
2. If not, decompose it into smaller relations in a desirable normal form
3. Decomposition must be **lossless** (no information loss)
4. Use **functional dependencies (FDs)** to guide the decomposition

6.1 Functional Dependencies

Functional Dependency (FD) — Definition

A functional dependency $\alpha \to \beta$ holds on a relation R if, for any two tuples t_1 and t_2 in R, whenever $t_1[\alpha] = t_2[\alpha]$, it must also be the case that $t_1[\beta] = t_2[\beta]$.
Key points:

- α determines β uniquely.
- If α is a superkey, then $\alpha \to R$ holds.
- Trivial FDs: $\beta \subseteq \alpha$, e.g., $A \to A$.

Example 221:

Consider $r(A, B)$ with the following instance of r. On this instance, $B \to A$ holds; $A \to B$ does NOT hold,

A	B
1	4
1	5
3	7

Example 222: Sample Relation

Consider the following instance of relation $r(A, B, C, D)$:

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Step-by-step FD analysis:

- Check if $A \to C$ holds:
  - Tuples with $A = a1$: $c1, c1 \to$ same $\to$ OK
  - Tuples with $A = a2$: $c2, c2 \to$ same $\to$ OK
  - $A = a3$: only one tuple $\to$ trivially OK
  - Conclusion: $A \to C$ **holds**
- Check if $C \to A$ holds:
  - Tuples with $C = c2$: $A = a2, a2, a3 \to$ different $\to$ violates FD
  - Conclusion: $C \to A$ **does not hold**

- Check if $B \to D$ holds:
  - $B = b2$: tuples $(a1, b2, c1, d2)$ and $(a2, b2, c2, d2)$ $\to$ D = d2 in both $\to$ holds
  - $B = b3$: tuples $(a2, b3, c2, d3)$ and $(a3, b3, c2, d4)$ $\to$ D = d3, d4 $\to$ violates FD
  - Conclusion: $B \to D$ **does not hold**
- Trivial FDs:
  - $A \to A$, $B \to B$, $C \to C$, $D \to D$ always hold
  - Any FD $\alpha \to \beta$ where $\beta \subseteq \alpha$ is trivial

Summary:

- Non-trivial FD that holds: $A \to C$
- Non-trivial FDs that do not hold: $C \to A$, $B \to D$

Example 223: Finding FDs — Student Table

sNo	sName	dNo	dName
101	Alice	D01	CS
102	Bob	D02	EE
103	Charlie	D01	CS
104	David	D03	ME

Analysis:

- sNo uniquely identifies student name and department $\to$ FD: $sNo \to sName, dNo$.
- dNo uniquely identifies department name $\to$ FD: $dNo \to dName$.
- Combination sNo, dNo is a superkey $\to$ $sNo, dNo \to sName, dName$.

Example 224: Employee Table FDs

empID	empName	dept	salary
E01	John	Sales	50000
E02	Mary	HR	55000
E03	Sam	Sales	52000
E04	Ann	IT	60000

Analysis:

- empID is unique $\to$ $empID \to empName, dept, salary$
- dept does not determine salary (Sales has multiple salaries) $\to$ no FD $dept \to salary$
- Combination empID, dept $\to$ still determines all attributes (superkey)

Example 225: Course Table FDs

cNo	cName	credits
C01	Math	4
C02	Physics	3
C03	CS	3
C04	Math	4

Analysis:

- cNo uniquely identifies course $\rightarrow cNo \rightarrow cName, credits$
- cName does not determine credits (Math appears twice, same credits here, but could vary in other instances)

Example 226: Register Table FDs

sNo	cNo	grade
101	C01	A
101	C02	B
102	C01	B
103	C03	A

Analysis:

- Combination (sNo, cNo) determines grade $\rightarrow (sNo, cNo) \rightarrow grade$
- sNo alone does not determine grade (student has multiple courses)
- cNo alone does not determine grade (multiple students take the same course)

Example 227: Classroom Table FDs

building	roomNo	capacity	type
A	101	50	Lecture
A	102	30	Lab
B	101	40	Lecture
B	102	20	Lab

Analysis:

- (building, roomNo) uniquely identifies capacity and type $\rightarrow (building, roomNo) \rightarrow capacity, type$
- roomNo alone does not determine capacity (room 101 exists in both buildings with different capacities)

Example 228: Composite Keys and FDs

orderID	productID	quantity	price
O01	P01	10	100
O01	P02	5	50
O02	P01	8	80

Analysis:

- Composite key (orderID, productID) $\rightarrow$ quantity, price
- orderID alone does not determine quantity or price
- productID alone does not determine quantity or price

FDs — Key Takeaways

- Identify superkeys: sets of attributes that uniquely identify tuples.
- Determine FDs by checking if a value of attribute(s) consistently determines other attribute(s) across all tuples.
- Trivial FDs always hold: $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$.
- Non-trivial FDs indicate constraints and guide normalization.
- Use tables to verify candidate keys and dependent attributes.

6.1.1 Types of Functional Dependency

Trivial Functional Dependency

Every dependent in trivial functional dependency is a subset of the determinant. If Y is a subset of X, the functional relationship $X \rightarrow Y$ is referred to as trivial.

Example 229: Example — Employee Table

Employee_Id	Name	Age
1	Zayn	24
2	Phobe	34
3	Hikki	26
4	David	29

Given that the dependent Name is a subset of the determinant Employee_Id, Name, the functional dependency between {Employee_Id, Name} and {Name} is trivial. Additionally, Name, Age, and Employee_Id are also trivial.

Non-Trivial Functional Dependency

The non-trivial functional dependency is opposed to the trivial one. Formally, the dependent is not a subset of the determinant. If Y is not a subset of X, the relationship $X \rightarrow Y$ is non-trivial.

Example 230: Example — Employee Table

Employee_Id	Name	Age
1	Zayn	24
2	Phobe	34
3	Hikki	26
4	David	29
5	Phobe	24

Because Name (dependent) is not a subset of Employee_Id, there is a non-trivial functional dependency between Employee_Id and Name. The functional dependencies {Employee_Id, Name} → {Age} are also non-trivial.

Multi-Valued Functional Dependency

Attributes in the dependent set are not reliant on each other in multi-valued dependency. If there is no functional dependency between Y and Z, the relationship $X \to Y, Z$ is referred to as multi-valued functional dependency.

Example 231: Example — Employee Table

Employee_Id	Name	Age
1	Zayn	24
2	Phobe	34
3	Hikki	26
4	David	29
4	Phobe	24

Since the dependent attributes Name and Age are not functionally dependent, {Employee_Id} → {Name, Age} is a multi-valued functional dependency.

Transitive Functional Dependency

Given two functional dependencies $A \to B$ and $B \to C$, $A \to C$ must also exist according to the transitivity principle. In transitive functional dependency, the dependent is indirectly reliant on the determinant.

Example 232: Example — Employee Table

Employee_Id	Name	Department	Street Number
1	Zayn	CD	11
2	Phobe	AB	24
3	Hikki	CD	11
4	David	PQ	71
5	Phobe	LM	21

The relationships between Employee_Id, Department, and Street Number are valid. Consequently, {Employee_Id} and {Street Number} are both valid functional dependencies according to the axiom of transitivity.

6.2 Armstrong's axioms (Inference rules)

Armstrong's Axioms are a set of rules that, when applied repeatedly, generate a closure of functional dependencies.

Axiom of Reflexivity

If $B \subseteq A$, then $A \to B$. This property is trivial: every set of attributes functionally determines its subsets.

Example 233: Example — Reflexivity

Let $A = \{\text{EmpID, Name}\}$ and $B = \{\text{Name}\}$. Since $B \subseteq A$, by reflexivity:

$$\{\text{EmpID, Name}\} \to \{\text{Name}\}$$

is a valid functional dependency.

Axiom of Augmentation

If $A \to B$ holds and C is any attribute set, then $AC \to BC$ also holds. Adding attributes to both sides preserves the dependency.

Example 234: Example — Augmentation

If EmpID $\to$ Name, then by augmentation:

$$\{\text{EmpID, Dept}\} \to \{\text{Name, Dept}\}$$

also holds.

Axiom of Transitivity

If $A \to B$ and $B \to C$, then $A \to C$. This is similar to the transitive property in algebra.

Example 235: Example — Transitivity

If EmpID $\to$ Dept and Dept $\to$ Manager, then EmpID $\to$ Manager also holds.

Union Rule

If $A \to B$ and $A \to C$, then $A \to BC$.

Example 236: Example — Union

If EmpID $\to$ Name and EmpID $\to$ Dept, then EmpID $\to$ Name, Dept.

Composition Rule

If $A \to B$ and $X \to Y$, then $AX \to BY$.

Example 237: Example — Composition

If EmpID $\to$ Name and DeptID $\to$ DeptName, then {EmpID, DeptID} $\to$ {Name, DeptName}.

Decomposition Rule

If $A \to BC$, then $A \to B$ and $A \to C$.

Example 238: Example — Decomposition

If EmpID $\to$ {Name, Dept}, then EmpID $\to$ Name and EmpID $\to$ Dept.

Pseudo Transitivity

If $A \to B$ and $BC \to D$, then $AC \to D$.

Example 239: Example — Pseudo Transitivity

If EmpID $\to$ Dept and {Dept, Location} $\to$ Manager, then {EmpID, Location} $\to$ Manager.

Self Determination

Every attribute set functionally determines itself: $A \to A$.

Example 240: Example — Self Determination

EmpID $\to$ EmpID is always true.

Extensivity

If $AC \to A$ and $A \to B$, then $AC \to B$. Also leads to $AC \to BC$ and $ABC \to BC$.

Example 241: Example — Extensivity

If {EmpID, Dept} $\to$ EmpID and EmpID $\to$ Name, then {EmpID, Dept} $\to$ Name.

Example 242: Reflexivity — MCQ

When an FD is reflexive, Y is a subset of X and X refers to the set of attributes, then:

(A) $Y \to X$ holds

(B) $X \to Y$ holds

(C) $XY \to Z$ holds

(D) None of the above

Answer: B. $X \to Y$ holds
Explanation: When an FD is reflexive, $Y \subseteq X$, so $X \to Y$ always holds according to Armstrong's reflexivity axiom.

Example 243: Union — MCQ

If $P \to Q$ and $P \to R$, then which of these would be true?

(A) $P \to QR$

(B) $P \to Q$

(C) $P \to R$

(D) All of the above

Answer: D. All of the above
Explanation: By the union rule, if $P \to Q$ and $P \to R$, then $P \to QR$ holds. Individual dependencies also hold.

6.3 Closure of attributes & Finding Candidate Keys

6.3.1 Closure of an Attribute

Closure of an Attribute

The **closure** of an attribute (or attribute set) is the set of all attributes that can be functionally determined from it, given a set of functional dependencies.
Purpose:

- Determine candidate keys
- Test dependency preservation
- Normalize relations

Algorithm to Compute X^+ (Closure of X):

Algorithm 1 Corrected Closure Calculation

Input: A set F of functional dependencies on a relation schema R, and a set of attributes $X \subseteq R$.

1. $X^+ \leftarrow X$ // Initialize closure with X itself
2. **Repeat**
3. oldX$^+ \leftarrow X^+$ // Snapshot current set at the start of the loop
4. **For each** functional dependency $Y \to Z$ in F:
5. **If** $X^+ \supseteq Y$, **then** $X^+ \leftarrow X^+ \cup Z$ // Add new attributes
6. **End For**
7. **Until** $X^+ =$ oldX$^+$ // If no changes occurred in this pass, STOP
8. **Return** X^+

Example 244: Closure of Single Attribute

Consider relation $R(A, B, C, D, E, F)$ with FDs:

$$F : E \to A,\ E \to D,\ A \to C,\ A \to D,\ AE \to F,\ AG \to K$$

Find E^+:

$$\begin{aligned} E^+ &= E \\ &= EA \quad (\text{from } E \to A) \\ &= EAD \quad (\text{from } E \to D) \\ &= EADC \quad (\text{from } A \to C) \\ &= EADCF \quad (\text{from } AE \to F) \\ &= EADCF \quad (\text{from } AG \to K, \text{ not added since } AG \not\subseteq E^+) \end{aligned}$$

Thus, $E^+ = \{E, A, D, C, F\}$.

Example 245: Closure of Multiple Attributes

Consider relation $R(A, B, C, D, E, F, G)$ with FDs:

$$F : E \to A,\ E \to D,\ A \to C,\ A \to D,\ AE \to F,\ AG \to K,\ BC \to DE,\ D \to F,\ CF \to G$$

Compute closures:

Closure of attribute A:

$$\begin{aligned} A^+ &= \{A\} \\ &= \{A, B, C\} \quad (\text{Using } A \to BC) \\ &= \{A, B, C, D, E\} \quad (\text{Using } BC \to DE) \\ &= \{A, B, C, D, E, F\} \quad (\text{Using } D \to F) \\ &= \{A, B, C, D, E, F, G\} \quad (\text{Using } CF \to G) \end{aligned}$$

Closure of attribute D:

$$D^+ = \{D, F\} \quad (\text{Using } D \to F)$$

Closure of attribute set $\{B, C\}$:

$$\begin{aligned} \{B, C\}^+ &= \{B, C\} \\ &= \{B, C, D, E\} \quad (\text{Using } BC \to DE) \\ &= \{B, C, D, E, F\} \quad (\text{Using } D \to F) \\ &= \{B, C, D, E, F, G\} \quad (\text{Using } CF \to G) \end{aligned}$$

Thus, $\{B, C\}^+ = \{B, C, D, E, F, G\}$.

6.3.2 Finding Candidate Keys

Candidate Key — Definition and Steps

A **candidate key** is defined as:

- A **minimal set of attributes** that can uniquely identify each tuple in a relation.
- Equivalently, a **minimal superkey** is a candidate key.

Steps to find candidate keys:

1. **Step 1: Find essential attributes**

- Essential attributes are **not present on the RHS** of any functional dependency OR Attributes that do not appear in any dependency at all.
- Essential attributes **must be in every candidate key**.

2. **Step 2: Identify non-essential attributes**
   - Remaining attributes that appear on RHS of some FD.
   - If essential attributes **alone** can determine all non-essential attributes, they form the only candidate key.
   - Otherwise, combine essential attributes with some non-essential attributes to form all possible candidate keys.

No. of Super Keys

Let $|R| = n$ (total attributes) and CK be a Candidate Key of size k.

- **Case 1: Single Candidate Key**
  The number of superkeys is given by: $N = 2^{n-k}$
- **Case 2: Two Candidate Keys (K_1 and K_2)**
  Using the Principle of Inclusion-Exclusion:
  $$|S_1 \cup S_2| = |S_1| + |S_2| - |S_1 \cap S_2|$$
  In terms of attributes:
  $$N = 2^{n-|K_1|} + 2^{n-|K_2|} - 2^{n-|K_1 \cup K_2|}$$
- **Case 3: Three Candidate Keys (K_1, K_2, K_3)**
  Applying the Principle of Inclusion-Exclusion (PIE):
  $$N = \sum |S_i| - \sum |S_i \cap S_j| + |S_1 \cap S_2 \cap S_3|$$
  In terms of attributes:
  $$N = (2^{n-|K_1|} + 2^{n-|K_2|} + 2^{n-|K_3|}) - (2^{n-|K_1 \cup K_2|} + 2^{n-|K_2 \cup K_3|} + 2^{n-|K_1 \cup K_3|}) + 2^{n-|K_1 \cup K_2 \cup K_3|}$$
- **Special Case (All attributes are Essential)**
  If all n attributes are essential, then the only candidate key is the set of all attributes ($k = n$):
  $$N = 2^{n-n} = 2^0 = 1$$
  In this case, the only superkey is the Relation R itself.

Example 246: Finding Candidate Key — Simple Closure

Relation: $R(A, B, C, D, E, F)$
Functional Dependencies:
$$C \to F, \quad E \to A, \quad EC \to D, \quad A \to B$$

Step 1: Essential attributes

$$\text{Attributes not on RHS: } C, E \quad \Rightarrow \text{Essential attributes} = \{C, E\}$$

Step 2: Compute closure of essential attributes

$$\{CE\}^+ = \{C, E\} \xrightarrow{C \to F} \{C, E, F\} \xrightarrow{E \to A} \{A, C, E, F\} \xrightarrow{EC \to D} \{A, C, D, E, F\} \xrightarrow{A \to B} \{A, B, C, D, E, F\}$$

Conclusion: CE can determine all attributes.
Candidate Key: CE (only one)

Super Keys: Essential attributes = 2, Non-essential = 4

$$\text{Number of super keys} = 2^4 = 16$$

Example 247: Candidate Key — Another Example

Relation: $R(A, B, C, D, E)$
Functional Dependencies:

$$AB \to C, \quad C \to D, \quad B \to E$$

Step 1: Essential attributes

$$\text{Attributes not on RHS: } A, B \quad \Rightarrow \text{Essential attributes} = \{A, B\}$$

Step 2: Compute closure of essential attributes

$$\{AB\}^+ = \{A, B\} \xrightarrow{AB \to C} \{A, B, C\} \xrightarrow{C \to D} \{A, B, C, D\} \xrightarrow{B \to E} \{A, B, C, D, E\}$$

Conclusion: AB can determine all attributes.
Candidate Key: AB (only one)
Super Keys: Essential = 2, Non-essential = 3

$$\text{Number of super keys} = 2^3 = 8$$

Example 248: Candidate Key — Multiple Attributes

Relation: $R(E, F, G, H, I, J, K, L, M, N)$
Functional Dependencies:

$$\{E, F\} \to G, \quad F \to I, J, \quad \{E, H\} \to K, L, \quad K \to M, \quad L \to N$$

Step 1: Essential attributes

$$\text{Attributes not on RHS: } E, F, H \quad \Rightarrow \text{Essential attributes} = \{E, F, H\}$$

Step 2: Compute closure of essential attributes

$$\begin{aligned}\{EFH\}^+ = \{E, F, H\} &\xrightarrow{EF \to G} \{E, F, G, H\} \\ &\xrightarrow{F \to I, J} \{E, F, G, H, I, J\} \\ &\xrightarrow{EH \to K, L} \{E, F, G, H, I, J, K, L\} \\ &\xrightarrow{K \to M} \{E, F, G, H, I, J, K, L, M\} \\ &\xrightarrow{L \to N} \{E, F, G, H, I, J, K, L, M, N\}\end{aligned}$$

Candidate Key: EFH (only one)
Super Keys: Essential = 3, Non-essential = 7

$$\text{Number of super keys} = 2^7 = 128$$

Example 249: Candidate Key — Optional Attributes

Relation: $R(A, B, C, D, E, H)$
Functional Dependencies:

$$A \to B, \quad BC \to D, \quad E \to C, \quad D \to A$$

Essential attributes: E, H

Candidate Keys: All options containing essential attributes that determine remaining attributes:

$$\text{Candidate Keys: } AEH, BEH, DEH$$

Answer: Option (D)

6.4 Canonical Cover

6.4.1 Equivalence of Functional Dependencies

Equivalence of Functional Dependencies

Two sets of functional dependencies E and F are said to be **equivalent** if

$$E^+ = F^+$$

Equivalently,

$$E \equiv F$$

iff

- E covers F
- F covers E

Example 250:

Let

$$F = \{A \to C,\ AC \to D,\ E \to AD,\ E \to H\}$$

$$G = \{A \to CD,\ E \to AH\}$$

Step 1: Check F covers G
Check $A \to CD$
Compute A^+ using F

$$A^+ = \{A\}$$

Using $A \to C$

$$A^+ = \{A, C\}$$

Using $AC \to D$

$$A^+ = \{A, C, D\}$$

Hence

$$A \to CD$$

holds.

Check $E \to AH$
Compute E^+ using F

$$E^+ = \{E\}$$

Using $E \to AD$

$$E^+ = \{E, A, D\}$$

Using $A \to C$

$$E^+ = \{E, A, D, C\}$$

Using $E \to H$

$$E^+ = \{E, A, D, C, H\}$$

Hence

$$E \to AH$$

holds.

$$\therefore F \text{ covers } G$$

Step 2: Check G covers F
Check $A \to C$
Compute A^+ using G

$$A^+ = \{A\}$$

Using $A \to CD$

$$A^+ = \{A, C, D\}$$

Hence

$$A \to C$$

holds.

Check $AC \to D$
Since $A \to CD$

$$AC \to D$$

holds.

Check $E \to AD$
Compute E^+ using G

$$E^+ = \{E\}$$

Using $E \to AH$

$$E^+ = \{E, A, H\}$$

Using $A \to CD$

$$E^+ = \{E, A, H, C, D\}$$

Hence

$$E \to AD$$

holds.

Check $E \to H$
Directly given.

$\therefore G$ covers F

Conclusion

$$F \equiv G$$

Hence the two sets of functional dependencies are **equivalent**.

Example 251:

Let

$$F = \{a \to b,\ b \to c,\ c \to d\}$$

$$G = \{a \to bc,\ c \to d\}$$

Step 1: Check if F covers G

We check each FD in G using the rules in F:

- **Check** $a \to bc$**:** Compute a^+ using F: $a^+ = \{a, b, c, d\}$. Since $\{b, c\} \subseteq a^+$, $a \to bc$ is **Covered**.
- **Check** $c \to d$**:** Compute c^+ using F: $c^+ = \{c, d\}$. Since $d \in c^+$, $c \to d$ is **Covered**.

$\therefore F$ covers G.

Step 2: Check if G covers F

We check each FD in F using the rules in G:

- **Check** $a \to b$**:** Compute a^+ using G: $a^+ = \{a, b, c, d\}$. Since $b \in a^+$, $a \to b$ is **Covered**.
- **Check** $b \to c$**:** Compute b^+ using G: $b^+ = \{b\}$. Since $c \notin b^+$, $b \to c$ is **NOT Covered**.
- **Check** $c \to d$**:** Compute c^+ using G: $c^+ = \{c, d\}$. Since $d \in c^+$, $c \to d$ is **Covered**.

$\therefore G$ does not cover F (specifically due to $b \to c$).

Example 252:

Let

$$F = \{a \to b,\ ab \to c,\ d \to ac,\ d \to e\}$$

$$G = \{a \to bc,\ d \to ab\}$$

Step 1: Check whether F covers G
Check $a \to bc$
Compute a^+ using F

$$a^+ = \{a\}$$

Using $a \to b$

$$a^+ = \{a, b\}$$

Using $ab \to c$

$$a^+ = \{a, b, c\}$$

Hence

$$a \to bc$$

holds.

Check $d \to ab$
Compute d^+ using F

$$d^+ = \{d\}$$

Using $d \to ac$

$$d^+ = \{d, a, c\}$$

Using $a \to b$

$$d^+ = \{d, a, c, b\}$$

Using $d \to e$

$$d^+ = \{d, a, c, b, e\}$$

Hence

$$d \to ab$$

holds.

$$\therefore F \text{ covers } G$$

Step 2: Check whether G covers F
Check $a \to b$
Compute a^+ using G

$$a^+ = \{a\}$$

Using $a \to bc$

$$a^+ = \{a, b, c\}$$

Hence $a \to b$ holds.

Check $ab \to c$
Since $a \to bc$

$$ab \to c$$

holds.

Check $d \to ac$
Compute d^+ using G

$$d^+ = \{d\}$$

Using $d \to ab$

$$d^+ = \{d, a, b\}$$

Using $a \to bc$

$$d^+ = \{d, a, b, c\}$$

Hence $d \to ac$ holds.

Check $d \to e$
Compute d^+ using G

$$d^+ = \{d, a, b, c\}$$

Attribute e cannot be derived.
Hence

$$d \to e$$

does not hold.

$$\therefore G \text{ does not cover } F$$

Conclusion

$$F \not\equiv G$$

Hence the two FD sets are **not equivalent**.

Example 253:

Let

$$F = \{pq \to r,\ p \to q,\ s \to pq,\ s \to t\}$$

$$G = \{p \to qr,\ s \to pt\}$$

Step 1: Check whether F covers G
Check $p \to qr$
Compute p^+ using F

$$p^+ = \{p\}$$

Using $p \to q$

$$p^+ = \{p, q\}$$

Using $pq \to r$

$$p^+ = \{p, q, r\}$$

Hence

$$p \to qr$$

holds.

Check $s \to pt$
Compute s^+ using F

$$s^+ = \{s\}$$

Using $s \to pq$

$$s^+ = \{s, p, q\}$$

Using $pq \to r$

$$s^+ = \{s, p, q, r\}$$

Using $s \to t$

$$s^+ = \{s, p, q, r, t\}$$

Hence

$$s \to pt$$

holds.

$$\therefore F \text{ covers } G$$

Step 2: Check whether G covers F
Check $pq \to r$
Compute pq^+ using G

$$pq^+ = \{p, q\}$$

Using $p \to qr$

$$pq^+ = \{p, q, r\}$$

Hence

$$pq \to r$$

holds.

Check $p \to q$
Compute p^+ using G

$$p^+ = \{p, q, r\}$$

Hence it holds.

Check $s \to pq$
Compute s^+ using G

$$s^+ = \{s\}$$

Using $s \to pt$

$$s^+ = \{s, p, t\}$$

Using $p \to qr$

$$s^+ = \{s, p, t, q, r\}$$

Hence

$$s \to pq$$

holds.

Check $s \to t$
Directly given in G.

$$\therefore G \text{ covers } F$$

Conclusion

$$F \equiv G$$

Hence the two sets of functional dependencies are **equivalent**.

6.4.2 Minimal Cover of Functional Dependencies

Minimal Cover Procedure

The Minimal Cover for a set of FDs E is a set of FDs F that satisfies the property that every dependency in E is in the closure F^+, and F contains no redundancies.
Procedure:

Step 1: Ensure every functional dependency (FD) has only a single attribute on the right-hand side.

- **Example:** $A \to BC$ becomes $\{A \to B,\ A \to C\}$.

Step 2: Delete Redundant FDs

Check each FD against the remaining set to see if it can be inferred.

- **Action:** For an FD $A \to B$, compute A^+ using all other FDs in the set.
- **Decision:** If $B \in A^+$, the FD is **Redundant** and is deleted.

Step 3: Minimize Left Side (Extraneous Attributes)

For FDs with multiple attributes on the left (e.g., $AB \to C$), check if one attribute is a "passenger."

Testing if B is Extraneous in $AB \to C$:

Compare the closure of the survivor (A) **before** and **after** removing B from that specific FD.

Comparison Result	Final Decision Logic
Not Equal ($A^+_{bef} \neq A^+_{aft}$)	**Cannot be removed.** The closure changed, so B is necessary.
Equal ($A^+_{bef} = A^+_{aft}$)	**Check:** Does A^+ contain the attribute being removed (B)? • If **YES** $\implies$ B is **Extraneous** (Remove it). • If **NO** $\implies$ B is **Necessary** (Keep it).

Example 254: Finding Minimal Cover

Given Relation $R(A, B, C, D)$ and $FDs = \{A \to B, C \to B, D \to ABC, AC \to D\}$.

Step 1: Standard Form (Right-side Decomposition)

1) $A \to B$
2) $C \to B$
3) $D \to A$
4) $D \to B$
5) $D \to C$
6) $AC \to D$

Step 2: Delete Redundant FDs

We check each FD against the remaining set:

- **Check $A \to B$:** Compute A^+ from $\{2, 3, 4, 5, 6\}$. $A^+ = \{A\}$. $B \notin A^+$. (Keep)
- **Check $C \to B$:** Compute C^+ from $\{1, 3, 4, 5, 6\}$. $C^+ = \{C\}$. $B \notin C^+$. (Keep)

- **Check $D \to B$:** Compute D^+ from $\{1, 2, 3, 5, 6\}$. $D^+ = \{D, A, C\}$ (using 3 and 5), then $\{D, A, C, B\}$ (using 1 or 2). Since $B \in D^+$, **$D \to B$ is Redundant** and is deleted.
- **Check $D \to A$:** D^+ from $\{1, 2, 5, 6\}$. $D^+ = \{D, C, B\}$. $A \notin D^+$. (Keep)
- **Check $D \to C$:** D^+ from $\{1, 2, 3, 6\}$. $D^+ = \{D, A, B\}$. $C \notin D^+$. (Keep)

Step 3: Minimize Left Side (Extraneous Attributes)

Check $AC \to D$ to see if A or C is redundant.

Try removing A (Check if $C \to D$)	**Try removing C (Check if $A \to D$)**
Compute C^+ before removing A:	Compute A^+ before removing C:
$C^+ = \{C, B\}$	$A^+ = \{A, B\}$
Compute C^+ after removing A (using $C \to D$):	Compute A^+ after removing C (using $A \to D$):
$C^+ = \{C, D, A, B\}$	$A^+ = \{A, D, C, B\}$
Result: $C^+_{before} \neq C^+_{after}$	**Result:** $A^+_{before} \neq A^+_{after}$
Hence, A **cannot be removed**.	Hence, C **cannot be removed**.

Final Irreducible Set (Minimal Cover)

The resulting set of FDs is:

$$\boxed{\begin{array}{l} A \to B \\ C \to B \\ D \to AC \\ AC \to D \end{array}}$$

Example 255: Finding Canonical Cover

Given: $F = \{X \to YZ,\ Y \to Z,\ X \to Y,\ XY \to Z\}$

Step 1: Standard Form (Right-side Decomposition)

1) $X \to Y$
2) $X \to Z$
3) $Y \to Z$
4) $XY \to Z$

Step 2: Delete Redundant FDs

Check each FD against the remaining set to see if it can be inferred:

- **Check $X \to Y$:** X^+ from $\{2, 3, 4\}$. $X^+ = \{X, Z\}$. $Y \notin X^+$. (Keep)
- **Check $X \to Z$:** X^+ from $\{1, 3, 4\}$. $X^+ = \{X, Y\}$ (using $X \to Y$), then $\{X, Y, Z\}$ (using $Y \to Z$). Since $Z \in X^+$, $X \to Z$ **is Redundant** and is deleted.

- **Check $Y \to Z$:** Y^+ from $\{1,4\}$. $Y^+ = \{Y\}$. $Z \notin Y^+$. (Keep)
- **Check $XY \to Z$:** $(XY)^+$ from $\{1,3\}$. $(XY)^+ = \{X,Y,Z\}$ (using $X \to Y$ and $Y \to Z$). Since $Z \in (XY)^+$, $XY \to Z$ **is Redundant** and is deleted.

Updated Set: $\{X \to Y,\ Y \to Z\}$

Step 3: Minimize Left Side (Extraneous Attributes)

In the updated set $\{X \to Y,\ Y \to Z\}$, all dependencies have only a single attribute on the left side. No further minimization is required.

Final Irreducible Set (Minimal Cover)

The resulting set of FDs is:

$$\boxed{\begin{array}{l} X \to Y \\ Y \to Z \end{array}}$$

Example 256: Finding Canonical Cover

Given: $F = \{A \to BC,\ CD \to E,\ B \to D,\ E \to A\}$

Step 1: Standard Form (Right-side Decomposition)

1) $A \to B$
2) $A \to C$
3) $CD \to E$
4) $B \to D$
5) $E \to A$

Step 2: Delete Redundant FDs

We check each FD against the remaining set:

- **Check $A \to B$:** A^+ from $\{2,3,4,5\}$. $A^+ = \{A,C\}$. $B \notin A^+$. (Keep)
- **Check $A \to C$:** A^+ from $\{1,3,4,5\}$. $A^+ = \{A,B,D\}$. $C \notin A^+$. (Keep)
- **Check $B \to D$:** B^+ from $\{1,2,3,5\}$. $B^+ = \{B\}$. $D \notin B^+$. (Keep)
- **Check $E \to A$:** E^+ from $\{1,2,3,4\}$. $E^+ = \{E\}$. $A \notin E^+$. (Keep)
- **Check $CD \to E$:** CD^+ from $\{1,2,4,5\}$. $CD^+ = \{C,D\}$. $E \notin CD^+$. (Keep)

Current Set: $\{A \to B,\ A \to C,\ B \to D,\ E \to A,\ CD \to E\}$

Step 3: Minimize Left Side (Extraneous Attributes)

Check $CD \to E$ to see if C or D is redundant.

Try removing C (Check if $D \to E$)	**Try removing D (Check if $C \to E$)**
Compute D^+ before removing C:	Compute C^+ before removing D:
$D^+ = \{D\}$	$C^+ = \{C\}$
Compute D^+ after removing C:	Compute C^+ after removing D:
$D^+ = \{D\}$ (No FD starts with D)	$C^+ = \{C\}$ (No FD starts with C)
Result: $D^+_{before} = D^+_{after}$	**Result:** $C^+_{before} = C^+_{after}$
Note: Since $E \notin D^+$, C **cannot be removed**.	**Note:** Since $E \notin C^+$, D **cannot be removed**.

Final Irreducible Set (Minimal Cover)

The set remains unchanged as no redundancies were found.

$$\boxed{\begin{array}{l} A \to BC \\ CD \to E \\ B \to D \\ E \to A \end{array}}$$

Example 257: Finding Canonical Cover - Redundancy First

Given Relation $R(A, B, C, D)$ and $FDs = \{AB \to C,\ A \to C,\ C \to D,\ B \to D\}$

Step 1: Standard Form (Right-side Decomposition)

1) $AB \to C$
2) $A \to C$
3) $C \to D$
4) $B \to D$

Step 2: Delete Redundant FDs

Check each FD against the remaining set to see if it can be inferred:

- **Check $AB \to C$:** Compute $(AB)^+$ using $\{2, 3, 4\}$. $(AB)^+ = \{A, B, C, D\}$. Since $C \in (AB)^+$, $AB \to C$ **is Redundant** and is deleted.
- **Check $A \to C$:** Compute A^+ using $\{3, 4\}$. $A^+ = \{A\}$. $C \notin A^+$. (Keep)
- **Check $C \to D$:** Compute C^+ using $\{2, 4\}$. $C^+ = \{C\}$. $D \notin C^+$. (Keep)
- **Check $B \to D$:** Compute B^+ using $\{2, 3\}$. $B^+ = \{B\}$. $D \notin B^+$. (Keep)

Updated Set: $\{A \to C,\ C \to D,\ B \to D\}$

Step 3: Minimize Left Side (Extraneous Attributes)

We examine the FDs in the updated set for multi-attribute left sides:

- $A \to C$: Single attribute LHS. No check needed.
- $C \to D$: Single attribute LHS. No check needed.
- $B \to D$: Single attribute LHS. No check needed.

Note: Since the only multi-attribute FD ($AB \to C$) was removed in Step 2, no further minimization is required.

Final Irreducible Set (Minimal Cover)

The resulting set of FDs is:

$$\boxed{\begin{array}{l} A \to C \\ C \to D \\ B \to D \end{array}}$$

Example 258: Canonical Cover: Step 3 is Mandatory

Given Relation $R(A, B, C, D)$ and $F = \{A \to B,\ B \to C,\ AB \to D\}$.

Step 1: Standard Form (RHS Decomposition)

1) $A \to B$
2) $B \to C$
3) $AB \to D$

Step 2: Delete Redundant FDs

Check if any entire FD can be inferred from the others:

- **Check $AB \to D$:** $(AB)^+$ from $\{A \to B, B \to C\}$. $(AB)^+ = \{A, B, C\}$. Since $D \notin (AB)^+$, $AB \to D$ **is NOT Redundant**. It survives Step 2.

Step 3: Minimize Left Side (Extraneous Attributes)

We check $AB \to D$ to see if B is extraneous.

Try removing B (Check if $A \to D$ is sufficient)

1. Compute A^+ before removing B from $AB \to D$:

$A^+ = \{A, B, C, D\}$ (using $A \to B, B \to C$, and the original $AB \to D$)

2. Compute A^+ after removing B from $AB \to D$:

$A^+ = \{A, B, C, D\}$ (using $A \to B, B \to C$, and the new $A \to D$)

Result: $A^+_{before} = A^+_{after}$

The Check: Does A^+ ($\{A, B, C, D\}$) contain the RHS attribute D?

Verdict: Yes. Therefore, B is **Extraneous** and can be removed.

Note: After removing B, the FD $AB \to D$ simplifies to $A \to D$.

Final Irreducible Set (Minimal Cover)

$$\boxed{\begin{array}{l} A \to B \\ B \to C \\ A \to D \end{array}}$$

6.5 Decomposition

Decomposition — Definition

Decomposition is the process of breaking a relation R into two or more sub-relations $R_1, R_2, \ldots$.
Properties of good decomposition:

1. **Lossless Decomposition:** Joining the decomposed relations gives back the original relation; no information is lost.
2. **Dependency Preservation:** All functional dependencies in the original relation can be enforced by the decomposed relations.

6.5.1 Lossless/Lossy Join Decomposition

Lossless Join Decomposition

Consider a relation R which is decomposed into sub-relations $R_1, R_2, \ldots, R_n$.
This decomposition is called **lossless join decomposition** when the join of the sub-relations results in the same relation R that was decomposed.
For lossless join decomposition, we always have:

$$R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n = R$$

where $\bowtie$ is the natural join operator.

Example 259: Lossless Join Decomposition

Consider the relation $R(A, B, C)$:

A	B	C
1	2	1
2	5	3
3	3	3

Decompose R into $R_1(A, B)$ and $R_2(B, C)$:

$R_1(A, B)$:

A	B
1	2
2	5
3	3

$R_2(B, C)$:

B	C
2	1
5	3
3	3

Performing the natural join $R_1 \bowtie R_2$ gives:

A	B	C
1	2	1
2	5	3
3	3	3

This is the same as the original relation R, hence the decomposition is **lossless**.

Note on Lossless Join Decomposition

Lossless join decomposition is also known as **non-additive join decomposition** because no extraneous tuples appear after joining the sub-relations.

Lossy Join Decomposition

Consider a relation R which is decomposed into sub-relations $R_1, R_2, \ldots, R_n$.
This decomposition is called **lossy join decomposition** when the join of the sub-relations does not result in the same relation R.
For lossy join decomposition:

$$R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n \supseteq R$$

where $\bowtie$ is the natural join operator. Some extraneous tuples appear in the join result.

Example 260: Lossy Join Decomposition

Consider the relation $R(A, B, C)$:

A	B	C
1	2	1
2	5	3
3	3	3

Decompose R into $R_1(A, C)$ and $R_2(B, C)$:

$R_1(A, C)$:

A	C
1	1
2	3
3	3

$R_2(B, C)$:

B	C
2	1
5	3
3	3

Performing the natural join $R_1 \bowtie R_2$ gives:

A	B	C
1	2	1
2	5	3
2	3	3
3	5	3
3	3	3

This relation is not the same as R and contains extraneous tuples. Hence, the decomposition is **lossy**.

Note on Lossy Join Decomposition

Lossy join decomposition is also known as **careless decomposition** because extraneous tuples make identification of original tuples difficult.

Determining Whether Decomposition Is Lossless Or Lossy

Consider a relation R decomposed into two sub-relations R_1 and R_2:

1. **Condition 1:** Union of both sub-relations must contain all attributes of R:

$$R_1 \cup R_2 = R$$

2. **Condition 2:** Intersection of both sub-relations must not be null:

$$R_1 \cap R_2 \neq \emptyset$$

3. **Condition 3:** Intersection of both sub-relations must be a super key of R_1, R_2, or both:

$$R_1 \cap R_2 = \text{Super key of } R_1 \text{ or } R_2$$

If all three conditions are satisfied, the decomposition is lossless. If any condition fails, the decomposition is lossy.

Example 261: Lossy Decomposition Verification

Consider $R(A, B, C, D)$ with functional dependencies $A \rightarrow B$ and $C \rightarrow D$. Decompose into $R_1(A, B)$ and $R_2(C, D)$.

Solution:

- Condition-01: $R_1 \cup R_2 = \{A, B, C, D\} = R \rightarrow$ satisfied
- Condition-02: $R_1 \cap R_2 = \emptyset \rightarrow$ fails
- Condition-03: Not checked as Condition-02 failed

Hence, decomposition is **lossy**.

6.5.1.1 The Chase Algorithm

The Chase Algorithm

To determine if a decomposition $\{R_1, R_2, \ldots, R_n\}$ is lossless:

- **Initialize:** Build a tableau (Rows: R_i, Cols: Attributes).
- **Fill:** Use **real symbols** $(a, b, c, \ldots)$ if attribute is present in R_i; otherwise use **dummies** $(a_i, b_i, c_i, \ldots)$.
- **Equate:** If two rows match on an FD's **LHS**, equate their **RHS**.
  - *Note: Always promote a dummy to a real symbol if possible.*
- **Converge:** Stop when no more changes can be made.

Any row is all real symbols $\implies$ Lossless
Otherwise $\implies$ Lossy

Example 262: Lossless Case: Chain Dependency

Consider $R(A, B, C, D)$ with FDs: $A \to B, B \to C, C \to D$. Decomposed into $R_1(A, B), R_2(B, C), R_3(C, D)$.
Step 1: Initial Tableau

	A	**B**	**C**	**D**
$R_1(A, B)$	a	b	c_1	d_1
$R_2(B, C)$	a_2	b	c	d_2
$R_3(C, D)$	a_3	b_3	c	d

Step 2: Apply $B \to C$
Rows R_1 and R_2 match on B (b). Promote c_1 in R_1 to real symbol c.

R_1 (upd)	a	b	$\boldsymbol{c}$	d_1

Step 3: Apply $C \to D$
Now R_1 and R_3 match on C (c). Promote d_1 in R_1 to real symbol d.

R_1 (final)	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{c}$	$\boldsymbol{d}$

Row 1 is all real symbols $\implies$ **Lossless**.

Example 263: Lossy Case: Disconnected Attributes

Consider $R(A, B, C, D)$ with FDs: $A \to B, C \to D$. Decomposed into $R_1(A, B), R_2(B, C), R_3(C, D)$.
Step 1: Initial Tableau

	A	B	C	D
$R_1(A,B)$	a	b	c_1	d_1
$R_2(B,C)$	a_2	b	c	d_2
$R_3(C,D)$	a_3	b_3	c	d

Step 2: Apply $C \to D$
R_2 and R_3 match on C (c). Promote d_2 in R_2 to d. **Step 3: Apply** $A \to B$
No rows match on A. No further changes possible. Since no row is (a, b, c, d), it is **Lossy**.

Example 264: Lossless Case: Converging FDs

$R(A, B, C)$ with $A \to B, B \to A$. Decomposed into $R_1(A, C), R_2(B, C)$.
Initial Tableau:

	A	B	C
$R_1(A,C)$	a	b_1	c
$R_2(B,C)$	a_2	b	c

Execution: No rows match on A or B. However, if we added $C \to B$: Rows R_1, R_2 match on C. Promote $b_1 \to b$. Now R_1 is $(a, b, c) \implies$ **Lossless**.

6.5.2 Dependency Preserving Decomposition

Definition

A decomposition of a relation R into sub-relations $R_1, R_2, \ldots, R_n$ is called **dependency preserving** if all functional dependencies of R satisfy at least one of the following:

1. The dependency is fully contained in one of the sub-relations R_i, or
2. The dependency can be derived from the FDs of the sub-relations without performing any joins.

In other words, all constraints of the original relation can be enforced locally in the sub-relations.

Why Dependency Preservation Matters

- Ensures **data integrity** after decomposition.
- Avoids **expensive joins** for enforcing constraints.
- Desired in normalization (3NF or BCNF) to combine lossless join and dependency preservation.

Step-by-Step Method to Check Dependency Preservation

Suppose a relation R with functional dependencies F is decomposed into sub-relations $R_1, R_2, \ldots, R_n$. To check if the decomposition is dependency preserving:

1. Find the functional dependencies F_i that hold in each sub-relation R_i.
2. Compute $F' = F_1 \cup F_2 \cup \cdots \cup F_n$.
3. Compare F' with the original dependency set F:

- If all dependencies in F are in F' or derivable from F', the decomposition is **dependency preserving**.
- If any dependency in F is missing and cannot be derived, the decomposition is **not dependency preserving**.

Example 265: Simple Dependency Preserving Decomposition

Let $R(A, B, C, D)$ with FD set:

$$F = \{A \to B, A \to C, C \to D\}$$

Decompose R into:

$$R_1(A, B, C), \quad R_2(C, D)$$

Step 1: Find FDs in sub-relations

- $F_1 = \{A \to B, A \to C\}$ in R_1
- $F_2 = \{C \to D\}$ in R_2

Step 2: Combine FDs:

$$F' = F_1 \cup F_2 = \{A \to B, A \to C, C \to D\} = F$$

Conclusion: All original FDs are preserved. Hence, the decomposition is **dependency preserving**.

Example 266: Non-Dependency Preserving Decomposition

Let $R(P, Q, R, S)$ with FD set:

$$F = \{PQ \to R, R \to S, S \to P\}$$

Decompose R into:

$$R_1(P, Q, R), \quad R_2(R, S)$$

Step 1: Find FDs in sub-relations

- $F_1 = \{PQ \to R, R \to P\}$ in R_1
- $F_2 = \{R \to S\}$ in R_2

Step 2: Combine FDs:

$$F' = F_1 \cup F_2 = \{PQ \to R, R \to S, R \to P\}$$

Step 3: Check original FDs

- $PQ \to R \to$ present in F'
- $R \to S \to$ present in F'
- $S \to P \to$ **missing**, cannot be derived from F'

Conclusion: Not all original FDs are preserved. Hence, the decomposition is **not dependency preserving**.

Example 267: Dependency Preserving Decomposition - Example 3

Consider relation $R(A, B, C, D, E)$ with functional dependencies:

$$F = \{A \to B, B \to C, C \to D, D \to E\}$$

Decompose R into:

$$R_1(A, B, C), \quad R_2(C, D, E)$$

Step 1: Find FDs in sub-relations

- $F_1 = \{A \to B, B \to C\}$ in R_1
- $F_2 = \{C \to D, D \to E\}$ in R_2

Step 2: Combine FDs:

$$F' = F_1 \cup F_2 = \{A \to B, B \to C, C \to D, D \to E\} = F$$

Step 3: Check all original FDs: All are present in F'.
Conclusion: The decomposition is **dependency preserving**.

Example 268: Non-Dependency Preserving Decomposition - Example 4

Consider relation $R(P, Q, R, S, T)$ with functional dependencies:

$$F = \{P \to Q, Q \to R, R \to S, S \to T, T \to P\}$$

Decompose R into:

$$R_1(P, Q, R), \quad R_2(R, S, T)$$

Step 1: Find FDs in sub-relations

- $F_1 = \{P \to Q, Q \to R\}$ in R_1
- $F_2 = \{R \to S, S \to T\}$ in R_2

Step 2: Combine FDs:

$$F' = F_1 \cup F_2 = \{P \to Q, Q \to R, R \to S, S \to T\}$$

Step 3: Check original FDs: The dependency $T \to P$ is missing and cannot be derived from F'.
Conclusion: The decomposition is **not dependency preserving**.

6.6 Normal Forms

Introduction to Normalization

The main reason for normalizing relations is to remove anomalies in the database. Failure to eliminate anomalies can lead to data redundancy, data inconsistency, and other integrity problems as the database grows. Normalization consists of a series of guidelines that help in creating a well-structured database.

Data Modification Anomalies

Data modification anomalies can be categorized into three types:

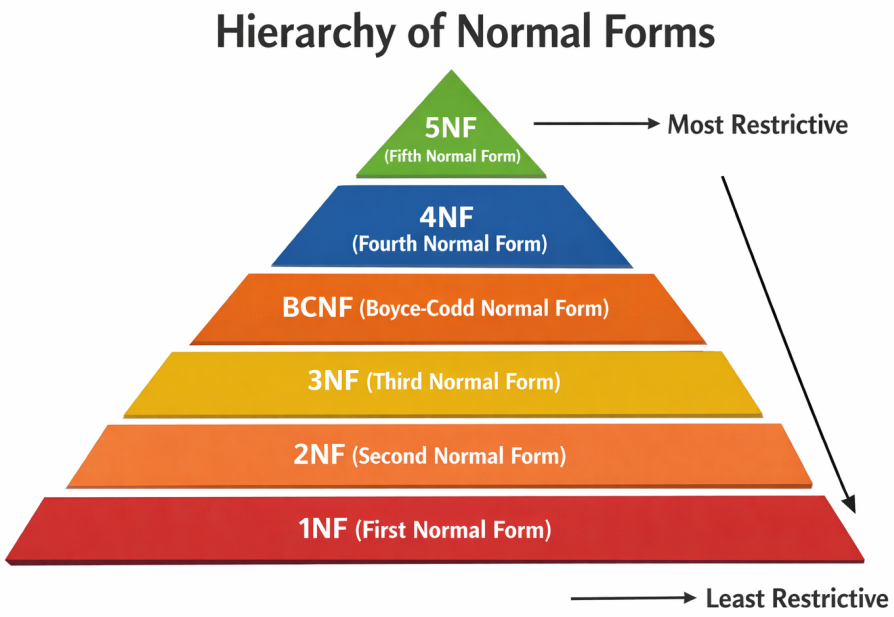
- **Insertion Anomaly:** Occurs when one cannot insert a new tuple into a relation due to lack of required data.
- **Deletion Anomaly:** Occurs when the deletion of data unintentionally results in loss of other important data.
- **Update Anomaly:** Occurs when updating a single data value requires multiple rows to be updated.

Definitions of Normal Forms

The standard normal forms in database design are defined as follows:

- **First Normal Form (1NF):** A relation is in 1NF if all attributes contain atomic (indivisible) values.
- **Second Normal Form (2NF):** A relation is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the candidate key.
- **Third Normal Form (3NF):** A relation is in 3NF if it is in 2NF and has no transitive dependencies (i.e., non-key attributes depend only on candidate keys).
- **Boyce-Codd Normal Form (BCNF):** A relation is in BCNF if it is in 3NF and for every functional dependency $X \to Y$, X is a superkey.

- **Fourth Normal Form (4NF):** A relation is in 4NF if it is in BCNF and does not contain any multi-valued dependencies.
- **Fifth Normal Form (5NF or Project-Join Normal Form):** A relation is in 5NF if it is in 4NF and every join dependency is implied by the candidate keys.



6.6.1 First Normal Form (1NF)

Concept of 1NF

A relation is said to be in First Normal Form (1NF) if:

1. Each attribute of every tuple contains only an **atomic value**, i.e., indivisible value.
2. Multi-valued attributes are not allowed; a single cell cannot contain a set, list, or nested table.
3. Attributes can be null, but must not be non-atomic (e.g., a list of values).

OR

A relation is in 1NF if the value of every attribute in each tuple is either single-valued or null.

Notes on 1NF

- By default, every relation in a relational database is considered to be in 1NF, because atomicity is part of the definition of a relation.
- 1NF is the **first step in normalization**, and all higher normal forms assume the relation is already in 1NF.

Example 269: 1NF - Step-by-Step

Problem: Check if the following relation is in 1NF and convert it if necessary.

Student_ID	Name	Subjects
100	Akshay	Computer Networks, Designing
101	Aman	Database Management Systems
102	Anjali	Automata, Compiler Design

Analysis:

- The **Subjects** attribute contains multiple values for some students.
- This violates the atomicity requirement of 1NF.

Solution: Convert to 1NF Rewrite the relation so that each cell contains a single value:

Student_ID	Name	Subjects
100	Akshay	Computer Networks
100	Akshay	Designing
101	Aman	Database Management Systems
102	Anjali	Automata
102	Anjali	Compiler Design

Conclusion: The above relation is now in 1NF because each attribute contains atomic values only.

Example 270: 1NF

Problem: Consider a relation storing employee skills:

Emp_ID	Name	Skills
201	Ravi	Java, Python
202	Meera	SQL
203	Sanjay	C++, Java

Analysis:

- The **Skills** attribute contains multiple values for some employees.
- This violates the 1NF requirement of atomic values.

Solution: Convert to 1NF

Emp_ID	Name	Skills
201	Ravi	Java
201	Ravi	Python
202	Meera	SQL
203	Sanjay	C++
203	Sanjay	Java

Conclusion: Each cell now contains a single atomic value, so the relation is in 1NF.

6.6.2 Second Normal Form (2NF)

Partial Dependency

A partial dependency occurs when a non-prime attribute is functionally dependent on a part (subset) of a candidate key rather than the whole key.
Formal Definition: Let $A \to B$ be a functional dependency in relation R. It is a partial dependency if:

- A is a proper subset of some candidate key, and
- B is a non-prime attribute.

Note: - To avoid partial dependency, an incomplete candidate key must not determine any non-prime attribute. - An incomplete candidate key can determine prime attributes without violating 2NF.

Definition of 2NF

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

1. R is already in 1NF.
2. There are no partial dependencies in R.

Procedure to Decompose a Relation into 2NF

Steps to Achieve 2NF

To decompose a relation R into Second Normal Form (2NF), follow these steps:

1. **Ensure 1NF:** Verify that R is already in First Normal Form (1NF), i.e., all attributes contain atomic values.
2. **Identify Candidate Keys:** Determine all candidate keys of R. Identify **prime attributes** (attributes in any candidate key) and **non-prime attributes** (attributes not in any candidate key).
3. **Identify Partial Dependencies:** Find all functional dependencies where a non-prime attribute is dependent on a part (subset) of a candidate key. These are partial dependencies.
4. **Create New Relations for Partial Dependencies:** For each partial dependency $X \to Y$, create a separate relation containing:
   - The determinant X (subset of candidate key)
   - All attributes functionally dependent on X (Y)

5. **Retain Remaining Attributes:** Create a relation containing:
   - All candidate key attributes
   - Remaining non-prime attributes not included in the new relations
6. **Verify 2NF:** Check that in each resulting relation:
   - No non-prime attribute is partially dependent on any candidate key.
   - Each relation is in 1NF.

Result: The original relation R is now decomposed into a set of relations that are in 2NF.

Example 271: Procedure Illustration

Consider relation $R(A, B, C, D, E)$ with functional dependencies:

$$A \to B, \quad B \to E, \quad C \to D$$

Step 1: Identify Candidate Key
$(AC)^+ = \{A, C, B, E, D\}$. Candidate key = **AC**.
Prime attributes = $\{A, C\}$; Non-prime = $\{B, D, E\}$.
Step 2: Identify Partial Dependencies (Prime → Non-Prime)
- $A \to B$: **Partial Dependency** ($A \subset AC$). - $C \to D$: **Partial Dependency** ($C \subset AC$). - $B \to E$: **NOT** a partial dependency (Non-prime → Non-prime).
Step 3: Create New Relations for Partial Dependencies
- $R_1(A, B, E)$: Handles partial key A. E moves here because $B \to E$ must follow B. - $R_2(C, D)$: Handles partial key C.
Step 4: Retain the Full Candidate Key
- $R_3(A, C)$: To act as the bridge/link.
Conclusion: Decomposition $\{R_1, R_2, R_3\}$ is in 2NF.

Example 272: Relation already in 2NF

Consider the relation $R(V, W, X, Y, Z)$ with functional dependencies:

$$VW \to XY, \quad Y \to V, \quad WX \to YZ$$

Step 1: Identify Candidate Keys
Candidate keys = **VW, WX, WY**.
Prime attributes = $\{V, W, X, Y\}$; Non-prime = $\{Z\}$.
Step 2: Identify Partial Dependencies
Check if any non-prime attribute depends on a *proper subset* of a key:
- Z depends on WX (Full Key).
- No non-prime attribute depends on V, W, X, or Y individually.
Conclusion: No partial dependency exists. Relation is in 2NF.

Example 273: Non-Prime dependencies

Consider relation $R(A, B, C, D, E)$ with functional dependencies:

$$A \to B, \quad B \to E, \quad C \to D$$

Step 1: Analysis
Candidate key = **AC**. Prime = $\{A, C\}$; Non-prime = $\{B, D, E\}$.
Step 2: Identify Violations
- $A \to B$: **Partial Dependency** (A is prime, B is non-prime). - $B \to E$: **Transitive Dependency** (Non-prime → Non-prime). Not a 2NF violation, but must be preserved.

Step 3: Decomposition (Lossless & DP)
- $R_1(A, B, E)$ (Fixes $A \to B$ and preserves $B \to E$). - $R_2(C, D)$ (Fixes $C \to D$). - $R_3(A, C)$ (Bridge table for the Full Key).
Conclusion: Each sub-relation is now in 2NF.

Example 274: Complex Relation

Consider $R(A, B, C, D, E, F, G, H, I, J)$ with functional dependencies:

$$AB \to C, \quad AD \to GH, \quad BD \to EF, \quad A \to I, \quad H \to J$$

Step 1: Analysis
Candidate key = **ABD**.
Prime = $\{A, B, D\}$; Non-prime = $\{C, E, F, G, H, I, J\}$.
Step 2: Identify Partial Dependencies
- $AB \to C$, $AD \to GH$, $BD \to EF$, $A \to I$ are **Partial** ($AB, AD, BD, A \subset ABD$). - $H \to J$ is **Transitive** (Non-prime $\to$ Non-prime).
Step 3: Create New Relations
- $R_1(A, B, C)$ - $R_2(A, D, G, H, J)$: J moves here with H to preserve dependency $H \to J$. - $R_3(B, D, E, F)$ - $R_4(A, I)$ - $R_5(A, B, D)$: The Full Key Bridge.
Conclusion: This decomposition is Lossless, Dependency Preserving, and in 2NF.

6.6.3 Third Normal Form (3NF)

Transitive Dependency

A transitive dependency $A \to B$ exists if and only if:

- A is **not a super key**, and
- B is a **non-prime attribute**.

NOTE: Transitive dependency must not exist for non-prime attributes. Transitive dependency can exist for prime attributes.

Definition of 3NF

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

1. The relation is already in **2NF**, and
2. No non-prime attribute is transitively dependent on any candidate key.

Alternative Definition of 3NF

A relation R is in 3NF if for every non-trivial functional dependency $A \to B$:

- A is a **super key**, or
- B is a **prime attribute**.

Procedure to Decompose a Relation into 3NF

To decompose a relation into 3NF, follow these steps:

1. Identify all **candidate keys** of the relation.
2. Identify **prime attributes** (attributes that are part of any candidate key) and **non-prime attributes**.

3. Examine each functional dependency $A \to B$:
   - If A is not a super key and B is a non-prime attribute, it is a transitive dependency.
4. For each transitive dependency, create a separate relation containing the determinant and the dependent attribute(s).
5. Remove the transitive dependencies from the original relation.
6. Repeat until no relation has transitive dependencies, ensuring all resulting relations are in 3NF.

Example 275: 3NF - Stepwise Example 1: Relation already in 3NF

Consider relation $R(A, B, C, D, E)$ with functional dependencies:

$$A \to BC, \quad CD \to E, \quad B \to D, \quad E \to A$$

Step 1: Identify candidate keys
Check which attribute sets determine all other attributes:

Candidate keys: A, E, CD, BC

Step 2: Identify prime and non-prime attributes

- Prime attributes $= \{A, B, C, D, E\}$ (all part of some candidate key)
- Non-prime attributes $= \{\}$ (none)

Step 3: Check for transitive dependency
Transitive dependency exists if $A \to B$ (A not super key) and B is non-prime. Here:

- No non-prime attributes exist.
- Therefore, no transitive dependency exists.

Step 4: Decomposition
Not required, as the relation has no transitive dependencies.
Step 5: Remove transitive dependencies
Not applicable.
Step 6: Verify 3NF
All non-prime attributes (none) are directly dependent on candidate keys.
Conclusion: Relation is in 3NF.

Example 276: 3NF - Stepwise Example 2: Relation not in 3NF

Consider relation $R(A, B, C, D)$ with functional dependencies:

$$AB \to CD, \quad A \to B, \quad B \to C$$

Step 1: Identify candidate keys
- AB determines CD - A alone does not determine all attributes - B alone does not determine all attributes

Candidate key: AB

Step 2: Identify prime and non-prime attributes

- Prime attributes $= \{A, B\}$
- Non-prime attributes $= \{C, D\}$

Step 3: Check for transitive dependency

- $A \to B$ (A is subset of candidate key AB) and $B \to C$ - C is non-prime $\to$ transitive dependency exists ($A \to C$ indirectly)
Step 4: Decompose relation
Create separate relations for transitive dependencies:

- $R_1(A, B)$ (key A)
- $R_2(B, C, D)$ (key B)

Step 5: Remove transitive dependencies
- Now, C and D are directly dependent on B in R_2 - B is fully dependent on A in R_1
Step 6: Verify 3NF
- $R_1(A, B)$: A is super key $\to$ 3NF satisfied - $R_2(B, C, D)$: B is super key $\to$ 3NF satisfied
Conclusion: Original relation violated 3NF, decomposition produces 3NF relations.

Example 277: 3NF - Stepwise Example 3: Complex Relation

Consider relation $R(A, B, C, D, E)$ with functional dependencies:

$$A \to B, \quad B \to E, \quad C \to D$$

Step 1: Identify candidate keys
- A determines B, B determines E $\to$ A can determine B and E - C determines D - To cover all attributes: candidate keys $= A, C$
Step 2: Identify prime and non-prime attributes
- Prime attributes $= \{A, C\}$ - Non-prime attributes $= \{B, D, E\}$
Step 3: Check for transitive dependency
- $A \to B$ and $B \to E \to A \to E$ (transitive) - E is non-prime $\to$ violates 3NF
Step 4: Decompose relation
- $R_1(A, B, E)$ with key A - $R_2(C, D)$ with key C
Step 5: Remove transitive dependencies
- R_1: B and E are now fully dependent on A - R_2: no transitive dependencies
Step 6: Verify 3NF
- $R_1(A, B, E)$: A is super key $\to$ 3NF satisfied - $R_2(C, D)$: C is super key $\to$ 3NF satisfied
Conclusion: Decomposition successfully converts relation into 3NF.

6.6.4 Boyce-Codd Normal Form (BCNF)

A relation is in **BCNF** if and only if:

1. The relation is already in 3NF.
2. For every non-trivial functional dependency $A \to B$, A is a super key of the relation.

Procedure to Decompose a Relation into BCNF

To decompose a relation R into BCNF:

1. Identify all candidate keys of R.
2. Check each functional dependency $A \to B$:
   - If A is a super key, it satisfies BCNF.
   - If A is not a super key, BCNF is violated.
3. For each violating dependency $A \to B$, decompose R into:
   - $R_1(A, B)$ — containing the violating dependency.

- $R_2(R - B)$ — the remaining attributes.

4. Repeat steps 1–3 for all decomposed relations until all relations satisfy BCNF.

Example 278: BCNF - Example 1: Relation Already in BCNF

Consider a relation $R(A, B, C)$ with functional dependencies:

$$A \to B, \quad B \to C, \quad C \to A$$

Step 1: Identify candidate keys: A, B, C
Step 2: Check BCNF condition: Each determinant (A, B, C) is a candidate key.
Conclusion: The relation already satisfies BCNF.

Example 279: BCNF - Example 2: Relation Not in BCNF

Consider a relation $R(A, B, C)$ with functional dependencies:

$$AB \to C, \quad C \to B$$

Step 1: Identify candidate keys: $(AB)^+ = ABC$, $(AC)^+ = ABC$ Candidate keys: AB, AC Prime attributes: A, B, C Non-prime attributes: None
Step 2: Check BCNF condition:

- $AB \to C$ is fine (AB is a super key)
- $C \to B$ violates BCNF because C is not a super key

Step 3: Decompose violating relation:

- $R_1(C, B)$ with key C
- $R_2(A, C)$ with key AC

Step 4: Check BCNF for new relations: Both R_1 and R_2 now satisfy BCNF.
Conclusion: Original relation decomposed into BCNF:

$$R_1(C, B), \quad R_2(A, C)$$

Example 280: BCNF - Example 3: Real Data Decomposition

Original relation:

$R(A, B, C)$

A	B	C
a	1	x
b	2	y
c	2	z
c	3	w
d	3	w
e	3	w

Functional dependencies: $AB \to C, C \to B$

Step 1: Candidate keys: AB, AC
Step 2: BCNF Violation: $C \to B$ (C not a super key)
Step 3: Decomposition:

$$R_1(C, B) =$$

C	B
x	1
y	2
z	2
w	3

$$R_2(A, C) =$$

A	C
a	x
b	y
c	z
c	w
d	w
e	w

Step 4: Check BCNF: Both R_1 and R_2 satisfy BCNF.
Conclusion: Decomposition removes redundancy and ensures BCNF.

6.6.5 Multi-valued Dependency and Fourth Normal Form (4NF)

The **Fourth Normal Form (4NF)** is satisfied if and only if:

1. The relation is in **BCNF**.
2. The relation contains no non-trivial **multi-valued dependencies (MVDs)** other than a candidate key.

Multi-valued Dependency (MVD)

A multi-valued dependency (MVD) in a relation R occurs when, for a single value of attribute A, there can be multiple independent values of attribute B, such that the presence of B values is independent of other attributes in R.
Notation: $A \twoheadrightarrow B$
Example: If a student can have multiple phone numbers and multiple courses, then:

$$\text{SID} \twoheadrightarrow \text{Phone}, \quad \text{SID} \twoheadrightarrow \text{Course}$$

Here, the phone numbers and courses are independent of each other, causing a multi-valued dependency.

Procedure to Decompose into 4NF

To decompose a relation R into 4NF:

1. Ensure that the relation is already in BCNF.

2. Identify all multi-valued dependencies $A \twoheadrightarrow B$.
3. For each MVD that violates 4NF (i.e., A is not a super key):
   - Create a new relation $R_1(A, B)$ for the MVD.
   - Create another relation $R_2(R - B)$ for the remaining attributes.
4. Repeat until all relations satisfy 4NF.

Example 281: 4NF - Example 1: Student-Course Database

Consider the following database tables:
Table R1 (Students):

SID	SNAME
S1	A
S2	B

Table R2 (Courses):

CID	CNAME
C1	C
C2	D

Step 1: Cross-product causes MVDs:

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

Step 2: Identify multi-valued dependencies:

- SID $\twoheadrightarrow$ CID
- SID $\twoheadrightarrow$ CNAME

Step 3: Check 4NF condition: SID is not a super key for the combined table. Hence, the relation violates 4NF.
Step 4: Decomposition into 4NF:

- $R_1(SID, CID)$
- $R_2(SID, CNAME)$
- $R_3(SID, SNAME)$ (to maintain student info)

Step 5: Check 4NF for new relations: All decomposed relations now satisfy 4NF.

6.6.6 Join Dependency and Fifth Normal Form (5NF)

A relation R is in **Fifth Normal Form (5NF)** if and only if:

1. The relation is already in **4NF**.
2. Every join dependency in R is implied by the candidate keys.
3. The relation cannot be further decomposed into smaller relations without losing the **lossless join property**.

Lossless Join Property

A decomposition of a relation is said to have a **lossless join** if, after decomposing the relation into two or more relations, performing a natural join on these decomposed relations reconstructs exactly the original relation without generating any spurious tuples.

Procedure to Decompose into 5NF

To check and decompose a relation R into 5NF:

1. Ensure that the relation is already in 4NF.
2. Identify all **join dependencies (JD)** in the relation.
3. For each join dependency $JD : R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n = R$:
   - Check if the decomposition is implied by the candidate keys.
   - If not, decompose R into smaller relations such that each decomposition satisfies the lossless join property.
4. Repeat until all join dependencies are satisfied by the candidate keys.

Example 282: 5NF - Example: Company-Agent-Product Schema

Consider the relation $ACP(Agent, Company, Product)$ representing which agent sells which product for which company.

Table ACP:

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

Step 1: Identify join dependency: The relation has a natural join dependency among:

$$(Agent, Company), \quad (Agent, Product), \quad (Company, Product)$$

Step 2: Decompose into 5NF:

- $R_1(Agent, Company)$

Agent	Company
A1	PQR
A1	XYZ
A2	PQR

- $R_2(Agent, Product)$

Agent	Product
A1	Nut
A1	Bolt
A2	Nut

- $R_3(Company, Product)$

Company	Product
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

Step 3: Check lossless join: The natural join of R_1, R_2, R_3 over the common attributes reconstructs the original table ACP exactly, without generating spurious tuples.
Step 4: Conclusion: All redundancies are eliminated, and the relation now satisfies 5NF. No further non-loss decompositions are possible.

6.7 Problems

Problem 182 *The relation $R(X, Y, Z, W)$ has the following instance:*

X	Y	Z	W
1	2	3	4
1	2	5	6
2	2	3	4

Which of the following FDs is NOT satisfied by this instance?

A. $X, Z \to W$

B. $Y, Z \to X$

C. $X, Y \to Z$

D. $W \to Z$

Problem 183 *Which of the following is a property of a prime attribute?*

A. *It cannot be part of a functional dependency.*

B. *It must be a member of at least one candidate key.*

C. *It must be a member of all candidate keys.*

D. *It cannot be NULL.*

Problem 184 *A relation R has 5 attributes $\{A, B, C, D, E\}$. How many possible non-trivial functional dependencies $X \to Y$ can be formed where X and Y are disjoint? (NAT)*

Problem 185 *Consider $R(A, B, C, D, E)$ with FDs $\{A \to B,\ BC \to D,\ D \to E\}$. The attribute C is:*

A. *Prime*

B. *Non-prime*

C. *Part of a candidate key*

D. *Both A and C*

Problem 186 *The FD $X \to Y$ is a partial dependency if:*

A. *X is a proper subset of a candidate key and Y is non-prime.*

B. *X is a candidate key and Y is non-prime.*

C. *X is a superkey and Y is prime.*

D. *X is a proper subset of a superkey.*

Problem 187 *Which of the following FDs can be derived from $F = \{A \to B,\ B \to C\}$ using Armstrong's Axioms? (MSQ)*

A. $A \to C$

B. $AD \to CD$

C. $AB \to C$

D. $C \to A$

Problem 188 *Consider $R(A, B, C, D, E)$ with $F = \{A \to B,\ B \to C,\ C \to D,\ D \to A\}$. How many attributes are non-prime? (NAT)*

Problem 189 *A relation $R(A, B, C)$ has no functional dependencies. What is the candidate key?*

A. $\{A\}$

B. $\{A, B, C\}$

C. *No candidate key exists.*

D. *Any single attribute.*

Problem 190 *Consider a relation instance $R(A, B, C, D)$:*

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_1	c_2	d_1
a_2	b_2	c_2	d_2

Which of the following functional dependencies are satisfied by this instance? (MSQ)

A. $A \to C$

B. $C \to A$

C. $B \to D$

D. $D \to B$

Problem 191 *Let $R(X, Y, Z)$ be a relation with the following instance:*

X	Y	Z
1	1	0
1	1	1
2	1	0

Which of the following is a candidate key for R?

A. $\{X, Y\}$

B. $\{X, Z\}$

C. $\{Y, Z\}$

D. $\{X, Y, Z\}$

Problem 192 *Consider a relation $R(A, B, C, D, E, F, G)$ with the following set of functional dependencies:*

$$F = \{AB \to C,\ C \to DE,\ E \to F,\ G \to A,\ G \to B\}$$

Which of the following is/are candidate key(s) for R? (MSQ)

A. $\{A, B, G\}$

B. $\{G\}$

C. $\{A, B\}$

D. $\{G, E\}$

Problem 193 *Let $Rel(P, Q, R, S, T, V)$ be a relational schema with the functional dependencies:*

$$P \to QR,\ RS \to T,\ QV \to P,\ T \to S$$

What is the total number of candidate keys for relation Rel? (NAT)

Problem 194 *How many functional dependencies are there in the closure F^+ for a relation $R(A, B)$ with a single dependency $A \to B$? (Exclude trivial dependencies like $A \to A$ and $AB \to A$, etc.) (NAT)*

Problem 195 *Consider a relation $R(A, B, C, D, E, H)$ with FDs:*

$$F = \{A \to B,\ BC \to D,\ E \to C,\ D \to A\}$$

Which of the following is NOT a candidate key?

A. AEH

B. BEH

C. CEH

D. DEH

Problem 196 *Consider a relation $R(A, B, C, D, E, F)$ with the following functional dependencies:*

$$F = \{AB \to C,\ C \to A,\ C \to D,\ D \to B,\ D \to E\}$$

Which of the following is/are candidate key(s) of R? (MSQ)

A. $\{A, B, F\}$

B. $\{C, F\}$

C. $\{D, F\}$

D. $\{A, D, F\}$

Problem 197 *Let $R(A, B, C, D, E)$ be a relation with the functional dependencies:*

$$A \to B,\ B \to C,\ C \to D,\ D \to E,\ E \to A$$

The number of superkeys for this relation is _____. (NAT)

Problem 198 *A relation $R(A, B, C, D, E)$ has A as the only candidate key. What is the maximum number of superkeys R can have? (NAT)*

Problem 199 *Consider $R(A, B, C, D, E, F)$ and $F = \{A \to BC,\ BC \to AD,\ D \to E,\ E \to D\}$. The number of candidate keys is _____. (NAT)*

Problem 200 *Let $R(A, B, C, D)$ be a relation where the only functional dependencies are:*

$$A \to B,\ B \to C,\ C \to A$$

How many super keys does R have? (NAT)

Problem 201 *Consider the following set of functional dependencies F over the schema $R(A, B, C, D, E)$:*

$$F = \{A \to BC,\ B \to D,\ D \to B,\ AB \to E\}$$

Which of the following is a canonical cover (minimal cover) for F?

A. $\{A \to C,\ A \to D,\ B \to D,\ D \to B,\ AB \to E\}$

B. $\{A \to B,\ A \to C,\ B \to D,\ D \to B,\ A \to E\}$

C. $\{A \to C,\ B \to D,\ D \to B,\ AB \to E\}$

D. $\{A \to B,\ B \to C,\ C \to D,\ D \to B,\ A \to E\}$

Problem 202 *Consider the set of functional dependencies $F = \{A \to B,\ B \to C,\ C \to D,\ D \to B\}$. Which of the following is/are a canonical cover F_c for F? (MSQ)*

A. $\{A \to B,\ B \to C,\ C \to D,\ D \to B\}$

B. $\{A \to B,\ B \to C,\ C \to D\}$

C. $\{A \to C,\ C \to D,\ D \to B\}$

D. $\{A \to B,\ B \to C,\ C \to B,\ C \to D\}$

Problem 203 *Let $F = \{A \to B,\ B \to C,\ AC \to D,\ D \to B\}$. The attribute C in $AC \to D$ is extraneous because:*

A. *$A \to B$ and $B \to C$ imply $A \to C$.*

B. *C is already on the LHS.*

C. *$B \to C$ is given.*

D. *$D \to B$ is given.*

Problem 204 *What is the canonical cover of $F = \{A \to B,\ AB \to C,\ A \to C\}$?*

A. $\{A \to B,\ B \to C\}$

B. $\{A \to B,\ A \to C\}$

C. $\{A \to BC\}$

D. $\{A \to B,\ AB \to C\}$

Problem 205 *Consider the decomposition of $R(A, B, C, D, E)$ into $R_1(A, B, C)$ and $R_2(C, D, E)$. If the FDs are $\{A \to B,\ C \to D\}$, the decomposition is:*

A. *Lossless and dependency preserving.*

B. *Lossy and dependency preserving.*

C. *Lossless and not dependency preserving.*

D. *Lossy and not dependency preserving.*

Problem 206 *A decomposition of R into $R_1, R_2, \ldots, R_n$ is dependency preserving if:*

A. $F = (F_1 \cup F_2 \cup \cdots \cup F_n)$

B. $F^+ = (F_1 \cup F_2 \cup \cdots \cup F_n)^+$

C. $(F_1 \cup F_2 \cup \cdots \cup F_n) \subseteq F$

D. $F \subseteq (F_1 \cup F_2 \cup \cdots \cup F_n)$

Problem 207 *Suppose a relation $R(A, B, C, D, E)$ is decomposed into $R_1(A, B, C)$ and $R_2(A, D, E)$. For the join to be lossless, which of the following must be true? (MSQ)*

A. *A must be a candidate key for R.*

B. *$A \to BC$ must be in F^+.*

C. *$A \to DE$ must be in F^+.*

D. $BC \to A$ *must be in* F^+.

Problem 208 *Let* $R(A, B, C)$ *be a relation. Under which of the following conditions is the decomposition of* R *into* $R_1(A, B)$ *and* $R_2(A, C)$ *always lossless?*

A. $B \to C$

B. $A \to B$

C. $BC \to A$

D. A *is a prime attribute.*

Problem 209 *In a relational schema* $R(A, B, C, D)$*, which of the following FDs is guaranteed to be preserved if we decompose* R *into* $R_1(A, B, C)$ *and* $R_2(A, D)$*?*

A. $B \to D$

B. $A \to D$

C. $BC \to D$

D. $D \to B$

Problem 210 *Given* $R(A, B, C, D)$ *and* $F = \{A \to B,\ B \to C,\ C \to D\}$*. If* R *is decomposed into* $R_1(A, C)$ *and* $R_2(B, D)$*, the dependency* $A \to B$ *is:*

A. *Preserved because* $A \in R_1$ *and* $B \in R_2$.

B. *Lost because no single decomposed relation contains both* A *and* B.

C. *Preserved because it can be inferred from other preserved dependencies.*

D. *Lost because the decomposition is lossy.*

Problem 211 *Let* $R(A, B, C, D)$ *be a relation schema with the set of functional dependencies* $F = \{A \to B,\ B \to C,\ C \to D\}$*. The relation is decomposed into* $R_1(A, B)$*,* $R_2(B, C)$*, and* $R_3(A, D)$*. Determine the properties of this decomposition:*

A. *The decomposition is lossless and dependency preserving.*

B. *The decomposition is lossy and dependency preserving.*

C. *The decomposition is lossless but not dependency preserving.*

D. *The decomposition is lossy and not dependency preserving.*

Problem 212 *Consider a relation* $R(P, Q, R, S, T)$ *with the functional dependencies* $F = \{P \to QR,\ RS \to T,\ Q \to S\}$*. The relation is decomposed into* $R_1(P, Q, R)$*,* $R_2(Q, S)$*, and* $R_3(R, S, T)$*. Which of the following statements is true regarding this decomposition?*

A. *It is lossless because* Q *is a candidate key for* R_2.

B. *It is lossy because no single row in the intersection table can be reduced to all* a*-symbols.*

C. *It is dependency preserving because all original FDs are contained within the sub-relations.*

D. *It is lossless and dependency preserving.*

Problem 213 *A relation* $R(A, B, C, D, E)$ *has the set of functional dependencies* $F = \{AB \to C,\ C \to D,\ D \to E,\ E \to A\}$*. The relation is decomposed into* $R_1(A, B, C)$*,* $R_2(C, D, E)$*, and* $R_3(E, A)$*. Testing the join and dependency properties reveals that the decomposition is:*

A. *Lossless and dependency preserving.*

B. *Lossy and dependency preserving.*

C. *Lossless and not dependency preserving.*

D. *Lossy and not dependency preserving.*

Problem 214 *Consider the relation $R(A, B, C, D, E)$ and the set of functional dependencies:*

$$F = \{A \to BC,\ CD \to E,\ B \to D,\ E \to A\}$$

Which of the following statements is/are TRUE? (MSQ)

A. *The relation R is in 3NF.*

B. *The attribute D is a prime attribute.*

C. *$E \to BC$ is a valid functional dependency in F^+.*

D. *R has exactly three candidate keys.*

Problem 215 *A relation $R(A, B, C, D, E, F)$ has the following functional dependencies:*

$$F = \{AB \to C,\ BC \to AD,\ D \to E,\ CF \to B\}$$

What is the highest normal form satisfied by relation R?

A. *1NF*

B. *2NF*

C. *3NF*

D. *BCNF*

Problem 216 *Which of the following statements regarding Normal Forms is/are FALSE? (MSQ)*

A. *If a relation is in BCNF and contains only one candidate key, it is always in 4NF.*

B. *Any relation with two attributes is always in BCNF.*

C. *A 3NF relation can have a transitive dependency if the dependent attribute is prime.*

D. *Every dependency-preserving decomposition into BCNF is also lossless.*

Problem 217 *For the relation $R(M, N, O, P, Q)$ with FDs:*

$$\{M \to N,\ N \to O,\ O \to M,\ MN \to P,\ OP \to Q\}$$

The relation R is in _____ normal form but not in _____ normal form.

A. *2NF, 3NF*

B. *3NF, BCNF*

C. *1NF, 2NF*

D. *BCNF, 4NF*

Problem 218 *A relation $R(A, B, C, D, E)$ is in 3NF but not in BCNF. Which of the following MUST be true?*

A. *There exists a partial dependency in R.*

B. *There exists a transitive dependency $X \to Y \to Z$ where Z is non-prime.*

C. *There exists a dependency $X \to A$ where A is a prime attribute and X is not a superkey.*

D. *R must have at least two overlapping candidate keys.*

Problem 219 *Which of the following statements is/are TRUE? (MSQ)*

A. *Every dependency-preserving decomposition into 3NF is also lossless.*

B. *A relation with no non-prime attributes is at least in 3NF.*

C. *BCNF is always dependency-preserving.*

D. *2NF eliminates partial dependencies.*

Problem 220 *A relation $R(A, B, C, D)$ is in BCNF. What is the minimum number of candidate keys R can have? (NAT)*

Problem 221 *If a relation is in BCNF, it is also in:*

A. *3NF*

B. *2NF*

C. *1NF*

D. *All of the above*

Problem 222 *Let $R(A, B, C, D)$ be a relation with $F = \{A \to B,\ B \to C,\ C \to D,\ D \to B\}$. Is R in 2NF?*

A. *Yes, because there are no partial dependencies.*

B. *No, because $A \to B$ is a partial dependency.*

C. *Yes, because it is in 3NF.*

D. *No, because it is not in 1NF.*

Problem 223 *Given the relation $R(A, B, C, D)$ and the set of FDs $F = \{AB \to C,\ C \to D,\ D \to A\}$. If we decompose R into $R_1(A, B, C)$ and $R_2(C, D)$:*

A. *R_1 is in BCNF and R_2 is in BCNF.*

B. *R_1 is in 3NF and R_2 is in BCNF.*

C. *R_1 is in 2NF and R_2 is in 3NF.*

D. *The decomposition is lossy.*

Problem 224 *Consider a relation $R(A, B, C)$ where the multivalued dependencies $A \twoheadrightarrow B$ and $A \twoheadrightarrow C$ hold. If the instance of R contains tuples (a_1, b_1, c_2) and (a_1, b_2, c_1), which of the following tuples MUST also exist in R for it to satisfy the MVDs? (MSQ)*

A. *(a_1, b_1, c_1)*

B. *(a_1, b_2, c_2)*

C. *(a_2, b_1, c_1)*

D. *No other tuples are required.*

Problem 225 *Consider a relation schema $R(A, B, C, D, E)$ with the following set of functional dependencies:*

$$F = \{AB \to C,\ A \to D,\ B \to E\}$$

*If R is decomposed into the minimum number of relations such that each resulting relation is in **Second Normal Form (2NF)**, how many tables will be in the final decomposition?*

Problem 226 *Consider a relation $R(A, B, C, D, E)$ with the functional dependencies:*

$$F = \{A \to B,\ B \to C,\ C \to D,\ D \to E\}$$

*Using the standard 3NF synthesis algorithm (Dependency Preserving and Lossless-Join), how many tables will the final decomposition of R contain to reach **Third Normal Form (3NF)**?*

Problem 227 *Let $R(A, B, C, D)$ be a relation schema with functional dependencies $F = \{A \to B,\ B \to C,\ C \to D,\ D \to B\}$. If the relation is decomposed into **Boyce-Codd Normal Form (BCNF)** using the standard decomposition algorithm, what is the minimum number of tables in the final lossless-join decomposition?*

Problem 228 *A relation $R(A, B, C, D)$ is in Fourth Normal Form (4NF) if:*

A. *It is in BCNF and contains no composite candidate keys.*

B. *It is in BCNF and every non-trivial multivalued dependency $X \twoheadrightarrow Y$ has X as a superkey.*

C. *It is in 3NF and all attributes are prime.*

D. *It is in BCNF and every join dependency is implied by the candidate keys.*

Problem 229 *Consider the relation $R(Student, Course, Hobby)$. A student can take multiple courses and engage in multiple hobbies. The courses and hobbies are independent of each other. Which of the following is TRUE?*

A. *The relation R has no multivalued dependencies.*

B. *R is in 4NF because there are no functional dependencies.*

C. *R violates 4NF because of the MVD $Student \twoheadrightarrow Course$.*

D. *R is in 5NF but not 4NF.*

Problem 230 *A relation R is in Fifth Normal Form (5NF), also known as Project-Join Normal Form (PJNF), if:*

A. *It is in 4NF and cannot be decomposed into any smaller relations.*

B. *It is in 4NF and every join dependency in R is implied by the candidate keys of R.*

C. *It is in BCNF and all multivalued dependencies are also functional dependencies.*

D. *It has only one candidate key which is a singleton set.*

Problem 231 *Which of the following is the strongest (most restrictive) normal form?*

A. *BCNF*

B. *3NF*

C. *4NF*

D. *5NF*

6.8 GATE PYQs

GATEPYQ 80 *Let $R(A, B, C, D, E)$ be a relational schema with functional dependency set*

$$F = \{A \to BC,\ CD \to E,\ E \to A\}.$$

Which of the following statements is correct? ***GATE DA 2026***

A. AD, ED and CD are the only candidate keys of R.
B. AD and ED are the only candidate keys of R.
C. A, E and CD are the only candidate keys of R.
D. A and CD are the only candidate keys of R.

GATEPYQ 81 *Let P, Q, R and S be the attributes of a relation in a relational schema. Let $X \to Y$ indicate functional dependency in the context of a relational database, where $X, Y \subseteq \{P, Q, R, S\}$. Which of the following options is/are always true?* ***GATE CSE 2026***

A. If $\{P, Q\} \to \{R\}$ and $\{P\} \to \{R\}$, then $\{Q\} \to \{R\}$
B. If $\{P, Q\} \to \{R\}$, then $\{P\} \to \{R\}$ or $\{Q\} \to \{R\}$
C. If $\{P\} \to \{R\}$ and $\{Q\} \to \{S\}$, then $\{P, Q\} \to \{R, S\}$
D. If $\{P\} \to \{R\}$, then $\{P, Q\} \to \{R\}$

GATEPYQ 82 *In the context of relational database normalization, which of the following statements is/are true?* ***GATE CSE 2026***

A. It is always possible to obtain a dependency-preserving 3NF decomposition of a relation
B. It is always possible to obtain a dependency-preserving 1NF decomposition of a relation
C. It is not always possible to obtain a dependency-preserving BCNF decomposition of a relation
D. It is not always possible to obtain a dependency-preserving 2NF decomposition of a relation

GATEPYQ 83 *Consider a relational database schema with a relation*

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

If $\{A, B\}$ and $\{A, C\}$ are the only two candidate keys of the relation R, then the number of superkeys of relation R is _____.

(Answer in integer)
GATE CSE 2026

GATEPYQ 84 *In the context of schema normalization in relational DBMS, consider a set F of functional dependencies. The set of all functional dependencies implied by F is called the closure of F. To compute the closure of F, Armstrong's Axioms can be applied. Consider X, Y, and Z as sets of attributes over a relational schema. The three rules of Armstrong's Axioms are described as follows.*

Reflexivity: *If $Y \subseteq X$, then $X \to Y$*
Augmentation: *If $X \to Y$, then $XZ \to YZ$ for any Z*
Transitivity: *If $X \to Y$ and $Y \to Z$, then $X \to Z$*
The additional rule of ***Union*** *is defined as follows.*
Union: *If $X \to Y$ and $X \to Z$, then $X \to YZ$*
It can be proved that the additional rule of Union is also implied by the three rules of Armstrong's Axioms. Listed below are four combinations of these three rules. Which one of these combinations is both necessary and sufficient for the proof?

GATE CSE 2026

A. Reflexivity, Augmentation, and Transitivity
B. Reflexivity and Augmentation
C. Transitivity
D. Augmentation and Transitivity

GATEPYQ 85 *Consider a database relation R with attributes*

$$R(A, B, C, D, E, F, G)$$

and having the following functional dependencies:

$$A \to BCEF, \quad D \to G, \quad E \to DG, \quad BC \to A$$

GATE DA 2025

Which of the following statements is/are correct?
A. A is the only candidate key of R
B. A, BC are the candidate keys of R
C. A, BC, E are the candidate keys of R
D. Relation R is not in Boyce-Codd Normal Form (BCNF)

GATEPYQ 86 *Given the relational schema*

$$R = (U, V, W, X, Y, Z)$$

and the set of functional dependencies:

$$\{U \to V,\ U \to W,\ WX \to Y,\ WX \to Z,\ V \to X\}$$

GATE DA 2024

Which of the following functional dependencies can be derived from the above set?
A. $VW \to YZ$
B. $WX \to YZ$
C. $VW \to U$
D. $VW \to Y$

GATEPYQ 87 *Consider a relational schema*

$$team(name, city, owner)$$

with functional dependencies:

$$\{name \to city,\ name \to owner\}$$

The relation team is decomposed into two relations:

$$t_1(name, city) \quad \textit{and} \quad t_2(name, owner)$$

Which of the following statement(s) is/are TRUE?
A. The relation team is NOT in BCNF.
B. The relations t_1 and t_2 are in BCNF.
C. The decomposition constitutes a lossless join.
D. The relation team is NOT in 3NF.
GATE CSE 2025

GATEPYQ 88 *Consider the following relational schemas along with all the functional dependencies that hold on them:*

$$R_1(A, B, C, D, E) : \{D \to E,\ EA \to B,\ EB \to C\}$$

$$R_2(A, B, C, D) : \{A \to D,\ A \to B,\ C \to A\}$$

Which of the following statement(s) is/are TRUE?
A. R_1 is in 3NF
B. R_2 is in 3NF
C. R_1 is NOT in 3NF
D. R_2 is NOT in 3NF
GATE CSE 2025

GATEPYQ 89 *A functional dependency $F : X \to Y$ is termed a useful functional dependency if and only if it satisfies all the following three conditions:*
1. X is not the empty set 2. Y is not the empty set 3. $X \cap Y = \emptyset$
For a relation R with 4 attributes, the total number of possible useful functional dependencies is
GATE CSE 2024

GATEPYQ 90 *The symbol $\rightarrow$ indicates functional dependency in the context of a relational database. Which of the following options is/are TRUE?*

A. $(X, Y) \rightarrow (Z, W)$ *implies* $X \rightarrow (Z, W)$
B. $(X, Y) \rightarrow (Z, W)$ *implies* $(X, Y) \rightarrow Z$
C. $((X, Y) \rightarrow Z$ *and* $W \rightarrow Y)$ *implies* $(X, W) \rightarrow Z$
D. $(X \rightarrow Y$ *and* $Y \rightarrow Z)$ *implies* $X \rightarrow Z$
GATE CSE 2024

GATEPYQ 91 *Which of the following statements about a relation R in first normal form (1NF) is/are TRUE?*

A. R *can have a multi-attribute key*
B. R *cannot have a foreign key*
C. R *cannot have a composite attribute*
D. R *cannot have more than one candidate key*
GATE CSE 2024

GATEPYQ 92 *Which one of the options given below refers to the degree (or arity) of a relation in relational database systems?*

A. Number of attributes of its relation schema
B. Number of tuples stored in the relation
C. Number of entries in the relation
D. Number of distinct domains of its relation schema
GATE CSE 2023

GATEPYQ 93 *Consider a relation $R(A, B, C, D, E)$ with the following functional dependencies:*

$$AB \rightarrow C, \quad BC \rightarrow D, \quad C \rightarrow E$$

The number of superkeys in the relation R is
GATE CSE 2022

GATEPYQ 94 *In a relational data model, which one of the following statements is TRUE?*

A. A relation with only two attributes is always in BCNF.
B. If all attributes of a relation are prime attributes, then the relation is in BCNF.
C. Every relation has at least one non-prime attribute.
D. BCNF decompositions preserve functional dependencies.
GATE CSE 2022

GATEPYQ 95 *Suppose the following functional dependencies hold on a relation U with attributes*

$$P, Q, R, S, T$$

$$P \rightarrow Q, \quad P \rightarrow R, \quad RS \rightarrow T$$

Which of the following functional dependencies can be inferred from the above functional dependencies?

A. $PS \rightarrow T$
B. $R \rightarrow T$
C. $P \rightarrow R$
D. $PS \rightarrow Q$
GATE CSE 2021

GATEPYQ 96 *Consider the relation $R(P, Q, S, T, X, Y, Z, W)$ with the following functional dependencies:*

$$PQ \rightarrow X, \quad P \rightarrow YX, \quad Q \rightarrow Y, \quad Y \rightarrow ZW$$

Consider the decomposition of the relation R into the constituent relations according to the following two decomposition schemes:

$$D_1 : R = [(P, Q, S, T); (P, T, X); (Q, Y); (Y, Z, W)]$$

$$D_2 : R = [(P, Q, S); (T, X); (Q, Y); (Y, Z, W)]$$

Which one of the following options is correct?
A. D_1 is a lossless decomposition, but D_2 is a lossy decomposition
B. D_1 is a lossy decomposition, but D_2 is a lossless decomposition
C. Both D_1 and D_2 are lossless decompositions
D. Both D_1 and D_2 are lossy decompositions
GATE CSE 2021

GATEPYQ 97 *Consider a relational table R that is in 3NF, but not in BCNF. Which one of the following statements is TRUE?*
A. R has a nontrivial functional dependency $X \rightarrow A$, where X is not a superkey and A is a prime attribute.
B. R has a nontrivial functional dependency $X \rightarrow A$, where X is not a superkey and A is a non-prime attribute and X is not a proper subset of any key.
C. R has a nontrivial functional dependency $X \rightarrow A$, where X is not a superkey and A is a non-prime attribute and X is a proper subset of some key.
D. A cell in R holds a set instead of an atomic value.
GATE CSE 2020

GATEPYQ 98 *Let the set of functional dependencies*

$$F = \{QR \rightarrow S,\ R \rightarrow P,\ S \rightarrow Q\}$$

hold on a relation schema

$$X = (P, Q, R, S)$$

X is not in BCNF. Suppose X is decomposed into two schemas

$$Y = (P, R) \quad \textit{and} \quad Z = (Q, R, S)$$

Consider the two statements given below:
I. Both Y and Z are in BCNF
II. Decomposition of X into Y and Z is dependency preserving and lossless
Which of the above statements is/are correct?
A. Both I and II
B. I only
C. II only
D. Neither I nor II
GATE CSE 2019

GATEPYQ 99 *Consider the following four relational schemas. For each schema, all non-trivial functional dependencies are listed. The underlined attributes are the respective primary keys.*

- *Schema I:* $Registration(\underline{rollno}, courses)$ *Non-trivial functional dependency:* $rollno \rightarrow courses$
- *Schema II:* $Registration(\underline{rollno, courseid}, email)$ *Non-trivial functional dependencies:* $rollno, courseid \rightarrow email$, $email \rightarrow rollno$
- *Schema III:* $Registration(\underline{rollno, courseid}, marks, grade)$ *Non-trivial functional dependencies:* $rollno, courseid \rightarrow marks, grade$, $marks \rightarrow grade$
- *Schema IV:* $Registration(\underline{rollno, courseid}, credit)$ *Non-trivial functional dependencies:* $rollno, courseid \rightarrow credit$, $courseid \rightarrow credit$

Which one of the relational schemas above is in 3NF but not in BCNF?
A. Schema I
B. Schema II
C. Schema III
D. Schema IV
GATE CSE 2018

GATEPYQ 100 *The following functional dependencies hold true for the relational schema*

$$R(V, W, X, Y, Z)$$

$$V \to W, \quad VW \to X, \quad Y \to VX, \quad Y \to Z$$

Which of the following is an irreducible equivalent for this set of functional dependencies?

A.

$$V \to W, \quad V \to X, \quad Y \to V, \quad Y \to Z$$

B.

$$V \to W, \quad W \to X, \quad Y \to V, \quad Y \to Z$$

C.

$$V \to W, \quad V \to X, \quad Y \to V, \quad Y \to X, \quad Y \to Z$$

D.

$$V \to W, \quad W \to X, \quad Y \to V, \quad Y \to X, \quad Y \to Z$$

GATE CSE 2017

GATEPYQ 101 *A database of research articles in a journal uses the following schema:*

$$(\textit{Volume}, \textit{Number}, \textit{StartPage}, \textit{EndPage}, \textit{Title}, \textit{Year}, \textit{Price})$$

The primary key is $(\textit{Volume}, \textit{Number}, \textit{StartPage}, \textit{EndPage})$ *and the following functional dependencies exist:*

$$(\textit{Volume}, \textit{Number}, \textit{StartPage}, \textit{EndPage}) \to \textit{Title}$$

$$(\textit{Volume}, \textit{Number}) \to \textit{Year}$$

$$(\textit{Volume}, \textit{Number}, \textit{StartPage}, \textit{EndPage}) \to \textit{Price}$$

The database is redesigned to use the following schemas:

$$(\textit{Volume}, \textit{Number}, \textit{StartPage}, \textit{EndPage}, \textit{Title}, \textit{Price})$$

$$(\textit{Volume}, \textit{Number}, \textit{Year})$$

Which is the weakest normal form that the new database satisfies, but the old one does not?

A. 1NF
B. 2NF
C. 3NF
D. BCNF

GATE CSE 2016

GATEPYQ 102 *Consider the relation* $X(P, Q, R, S, T, U)$ *with the following set of functional dependencies:*

$$F = \{\{P, R\} \to \{S, T\}, \ \{P, S, U\} \to \{Q, R\}\}$$

Which of the following is a trivial functional dependency in F^+*, where* F^+ *is the closure of* F*?*

A. $\{P, R\} \to \{S, T\}$
B. $\{P, R\} \to \{R, T\}$
C. $\{P, S\} \to \{S\}$
D. $\{P, S, U\} \to \{Q\}$

GATE CSE 2015

GATEPYQ 103 *A prime attribute of a relation schema R is an attribute that appears*
A. in all candidate keys of R.
B. in some candidate key of R.
C. in a foreign key of R.
D. only in the primary key of R.
GATE CSE 2014

GATEPYQ 104 *Given the following two statements:*
S1: Every table with two single-valued attributes is in 1NF, 2NF, 3NF and BCNF.
S2: $AB \rightarrow C,\ D \rightarrow E,\ E \rightarrow C$ is a minimal cover for the set of functional dependencies $AB \rightarrow C,\ D \rightarrow E,\ AB \rightarrow E,\ E \rightarrow C$.
Which one of the following is CORRECT?
A. S1 is TRUE and S2 is FALSE
B. Both S1 and S2 are TRUE
C. S1 is FALSE and S2 is TRUE
D. Both S1 and S2 are FALSE
GATE CSE 2014

GATEPYQ 105 *Consider the relation scheme $R = (E, F, G, H, I, J, K, L, M, N)$ and the set of functional dependencies:*

$$\{\{E, F\} \rightarrow \{G\},\ \{F\} \rightarrow \{I, J\},\ \{E, H\} \rightarrow \{K, L\},\ \{K\} \rightarrow \{M\},\ \{L\} \rightarrow \{N\}\}$$

What is the key for R?
A. $\{E, F\}$
B. $\{E, F, H\}$
C. $\{E, F, H, K, L\}$
D. $\{E\}$
GATE CSE 2014

GATEPYQ 106 *Relation R has eight attributes A, B, C, D, E, F, G, H. Fields of R contain only atomic values.*
Let

$$F = \{CH \rightarrow G,\ A \rightarrow BC,\ B \rightarrow CFH,\ E \rightarrow A,\ F \rightarrow EG\}$$

be the set of functional dependencies (FDs) such that F^+ is exactly the set of FDs that hold for R.
How many candidate keys does the relation R have?
A. 3
B. 4
C. 5
D. 6
GATE CSE 2013

GATEPYQ 107 *Relation R has eight attributes A, B, C, D, E, F, G, H. Fields of R contain only atomic values.*
Let

$$F = \{CH \rightarrow G,\ A \rightarrow BC,\ B \rightarrow CFH,\ E \rightarrow A,\ F \rightarrow EG\}$$

be the set of functional dependencies (FDs) such that F^+ is exactly the set of FDs that hold for R.
The relation R is
A. in 1NF, but not in 2NF
B. in 2NF, but not in 3NF
C. in 3NF, but not in BCNF
D. in BCNF
GATE CSE 2013

GATEPYQ 108 *Which of the following is TRUE?*
A. Every relation in 3NF is also in BCNF
B. A relation R is in 3NF if every non-prime attribute of R is fully functionally dependent on every key of R
C. Every relation in BCNF is also in 3NF
D. No relation can be in both BCNF and 3NF
GATE CSE 2012

GATEPYQ 109 *Consider the following relational schema:*

Suppliers(sid: integer, sname: string, city: string, street: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

Assume that, in the `Suppliers` *relation, each supplier and each street within a city has a unique name, and* (*sname*, *city*) *forms a candidate key. No other functional dependencies are implied other than those implied by primary and candidate keys.*

Which one of the following is TRUE about the above schema?
A. The schema is in BCNF
B. The schema is in 3NF but not in BCNF
C. The schema is in 2NF but not in 3NF
D. The schema is not in 2NF
GATE CSE 2009

GATEPYQ 110 *Consider the following relational schemes for a library database:*

Book(Title, Author, Catalog_no, Publisher, Year, Price)

Collection(Title, Author, Catalog_no)

with the following functional dependencies:
I. $Title, Author \rightarrow Catalog\_no$
II. $Catalog\_no \rightarrow Title, Author, Publisher, Year$
III. $Publisher, Title, Year \rightarrow Price$
Assume $\{Author, Title\}$ *is the key for both schemes.*
Which of the following statements is TRUE?
A. Both Book and Collection are in BCNF
B. Both Book and Collection are in 3NF only
C. Book is in 2NF and Collection is in 3NF
D. Both Book and Collection are in 2NF only
GATE CSE 2008

GATEPYQ 111 *Which one of the following statements is FALSE?*
A. Any relation with two attributes is in BCNF
B. A relation in which every key has only one attribute is in 2NF
C. A prime attribute can be transitively dependent on a key in a 3NF relation
D. A prime attribute can be transitively dependent on a key in a BCNF relation
GATE CSE 2007

GATEPYQ 112 *The following functional dependencies are given:*

$$AB \rightarrow CD,\ AF \rightarrow D,\ DE \rightarrow F,\ C \rightarrow G,\ F \rightarrow E,\ G \rightarrow A$$

Which one of the following options is FALSE?
A. $\{CF\}^+ = \{A, C, D, E, F, G\}$
B. $\{BG\}^+ = \{A, B, C, D, G\}$
C. $\{AF\}^+ = \{A, C, D, E, F, G\}$
D. $\{AB\}^+ = \{A, B, C, D, G\}$
GATE CSE 2006

GATEPYQ 113 *Which one of the following statements about normal forms is FALSE?*
A. BCNF is stricter than 3NF
B. Lossless, dependency-preserving decomposition into 3NF is always possible
C. Lossless, dependency-preserving decomposition into BCNF is always possible
D. Any relation with two attributes is in BCNF
GATE CSE 2005

GATEPYQ 114 *Consider a relation scheme $R = (A, B, C, D, E, H)$ on which the following functional dependencies hold:*

$$\{A \rightarrow B,\ BC \rightarrow D,\ E \rightarrow C,\ D \rightarrow A\}$$

What are the candidate keys of R?
A. AE, BE
B. AE, BE, DE
C. AEH, BEH, BCH
D. AEH, BEH, DEH
GATE CSE 2005

GATEPYQ 115 *The relation scheme StudentPerformance(name, courseNo, rollNo, grade) has the following functional dependencies:*

name, courseNo $\rightarrow$ grade
rollNo, courseNo $\rightarrow$ grade
name $\rightarrow$ rollNo
rollNo $\rightarrow$ name

The highest normal form of this relation scheme is
A. 2NF
B. 3NF
C. BCNF
D. 4NF
GATE CSE 2004

GATEPYQ 116 *Consider the following functional dependencies in a database:*

Date_of_Birth $\rightarrow$ Age	*Age $\rightarrow$ Eligibility*
Name $\rightarrow$ Roll_number	*Roll_number $\rightarrow$ Name*
Course_number $\rightarrow$ Course_name	*Course_number $\rightarrow$ Instructor*
(Roll_number, Course_number) $\rightarrow$ Grade	

The relation (Roll_number, Name, Date_of_birth, Age) is
A. in second normal form but not in third normal form
B. in third normal form but not in BCNF
C. in BCNF
D. in none of the above
GATE CSE 2003

GATEPYQ 117 *Relation R with an associated set of functional dependencies F is decomposed into BCNF. The redundancy (arising out of functional dependencies) in the resulting set of relations is*
A. Zero
B. More than zero but less than that of an equivalent 3NF decomposition
C. Proportional to the size of F^+
D. Indeterminate
GATE CSE 2002

GATEPYQ 118 *From the following instance of relation schema $R(A, B, C)$:*

A	B	C
1	1	1
1	1	0
2	3	2
2	3	2

We can conclude that:
A. A functionally determines B and B functionally determines C
B. A functionally determines B and B does not functionally determine C
C. B does not functionally determine C
D. A does not functionally determine B and B does not functionally determine C
GATE CSE 2002

GATEPYQ 119 *Let r be a relation instance with schema $R = (A, B, C, D)$. We define*

$$r_1 = \Pi_{A,B,C}(r) \quad \textit{and} \quad r_2 = \Pi_{A,D}(r)$$

Let

$$s = r_1 * r_2$$

where $*$ denotes natural join. Given that the decomposition of r into r_1 and r_2 is lossy, which one of the following is TRUE? ***GATE CSE 2005***
A. $s \subset r$
B. $r \cup s = r$
C. $r \subset s$
D. $r * s = s$

GATEPYQ 120 *$R(A, B, C, D)$ is a relation. Which of the following does not have a lossless-join, dependency-preserving BCNF decomposition?*
A. $A \rightarrow B,\ B \rightarrow CD$
B. $A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D$
C. $AB \rightarrow C,\ C \rightarrow AD$
D. $A \rightarrow BCD$
GATE CSE 2001

GATEPYQ 121 *Consider a schema $R(A, B, C, D)$ and the functional dependencies:*

$$A \rightarrow B, \quad C \rightarrow D$$

Then the decomposition of R into $R_1(AB)$ and $R_2(CD)$ is:
A. Dependency preserving and lossless join
B. Lossless join but not dependency preserving
C. Dependency preserving but not lossless join
D. Not dependency preserving and not lossless join
GATE CSE 2001

GATEPYQ 122 *Given the following relation instance:*

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

Which of the following functional dependencies are satisfied by the instance?
A. $XY \rightarrow Z$ *and* $Z \rightarrow Y$
B. $YZ \rightarrow X$ *and* $Y \rightarrow Z$
C. $YZ \rightarrow X$ *and* $X \rightarrow Z$
D. $XZ \rightarrow Y$ *and* $Y \rightarrow X$
GATE CSE 2000

GATEPYQ 123 *Consider the schema* $R = (S, T, U, V)$ *and the functional dependencies:*

$$S \rightarrow T, \quad T \rightarrow U, \quad U \rightarrow V, \quad V \rightarrow S$$

Let R *be decomposed into* $(R_1$ *and* $R_2)$ *such that* $R_1 \cap R_2 \neq \emptyset$.
The decomposition is
A. not in 2NF
B. in 2NF but not 3NF
C. in 3NF but not in 2NF
D. in both 2NF and 3NF
GATE CSE 1999

GATEPYQ 124 *Which normal form is considered adequate for normal relational database design?*
A. 2NF
B. 5NF
C. 4NF
D. 3NF
GATE CSE 1998

GATEPYQ 125 *For a database relation* $R(a, b, c, d)$*, where the domains* a, b, c, d *include only atomic values, only the following functional dependencies and those that can be inferred from them hold:*

$$a \rightarrow c, \quad b \rightarrow d$$

This relation is
A. in first normal form but not in second normal form
B. in second normal form but not in first normal form
C. in third normal form
D. none of the above
GATE CSE 1997

6.9 Try it Yourself

Exercise 177 *Consider a database relation $R(P, Q, R, S, T, U, V)$ with the following set of functional dependencies:*

$$P \to QRST, \quad U \to V, \quad S \to UV, \quad QR \to P$$

Which of the following statements is/are correct? (MSQ)

A. P *is the only candidate key of* R

B. P *and* QR *are the candidate keys of* R

C. P, QR, *and* S *are the candidate keys of* R

D. *Relation* R *is not in Boyce-Codd Normal Form (BCNF)*

Exercise 178 *Given the relational schema $R(A, B, C, D, E, F)$ and the set of functional dependencies:*

$$A \to B, \quad A \to C, \quad CD \to E, \quad CD \to F, \quad B \to D$$

Which of the following functional dependencies can be inferred from the above set? (MSQ)

A. $BC \to EF$

B. $CD \to EF$

C. $BC \to A$

D. $BC \to E$

Exercise 179 *Consider a relational schema $Course(cid, dept, HOD)$ with functional dependencies:*

$$cid \to dept, \quad cid \to HOD$$

The relation is decomposed into:

$$C_1(cid, dept) \quad \text{and} \quad C_2(cid, HOD)$$

Which of the following statement(s) is/are TRUE? (MSQ)

A. *The relation* $Course$ *is in BCNF.*

B. *The relations* C_1 *and* C_2 *are in BCNF.*

C. *The decomposition is a lossless join.*

D. *The relation* $Course$ *satisfies 3NF.*

Exercise 180 *Consider the following two relational schemas and their functional dependencies:*

$$R_1(X, Y, Z, W, P) : \{W \to P,\ PX \to Y,\ PY \to Z\}$$

$$R_2(X, Y, Z, W) : \{X \to W,\ X \to Y,\ Z \to X\}$$

Which of the following statement(s) is/are TRUE?

A. R_1 *is in 3NF*

B. R_2 *is in 3NF*

C. R_1 *is NOT in 3NF*

D. R_2 *is NOT in 3NF*

Exercise 181 *A functional dependency $X \to Y$ is defined as **proper** if $X \neq \emptyset, Y \neq \emptyset$, and $X \cap Y = \emptyset$. For a relational schema with 3 attributes, the total number of such proper functional dependencies is _____. (NAT)*

Exercise 182 *Let $\to$ denote functional dependency. Which of the following is/are TRUE? (MSQ)*

A. $X \to YZ$ *implies* $X \to Y$ *and* $X \to Z$

B. $XY \to Z$ *implies* $X \to Z$

C. $(X \to Y$ *and* $WY \to Z)$ *implies* $WX \to Z$

D. $X \to Y$ *implies* $XZ \to YZ$

Exercise 183 *Which of the following statements about a relation* R *in First Normal Form (1NF) is/are TRUE? (MSQ)*

A. R *must have a single-attribute primary key.*

B. R *can contain multiple candidate keys.*

C. R *forbids the use of nested relations as attribute values.*

D. R *must not have any transitive dependencies.*

Exercise 184 *The **arity** of a relation in the relational model refers to:*

A. *The total number of tuples in the relation.*

B. *The number of columns in the relation schema.*

C. *The number of distinct values in a specific column.*

D. *The cardinality of the domain of the primary key.*

Exercise 185 *Consider a relation* $R(P, Q, R, S, T)$ *with the following functional dependencies:*

$$PQ \to R, \quad QR \to S, \quad R \to T$$

The total number of superkeys for relation R *is _____. (NAT)*

Exercise 186 *Which of the following is TRUE in the context of normalization?*

A. *Every relation with exactly two attributes is in BCNF.*

B. *If every attribute of* R *is a member of some candidate key,* R *is in BCNF.*

C. *3NF is stricter than BCNF.*

D. *A 3NF decomposition always guarantees the preservation of all dependencies.*

Exercise 187 *Suppose the following FDs hold on* $R(A, B, C, D, E)$*:*

$$A \to B, \quad A \to C, \quad CD \to E$$

Which of the following is/are implied? (MSQ)

A. $AD \to E$

B. $C \to E$

C. $AC \to B$

D. $AD \to B$

Exercise 188 *Consider relation* $R(A, B, C, D, E, F, G, H)$ *with:*

$$AB \to E, \quad A \to FE, \quad B \to F, \quad F \to GH$$

Decomposition schemes:

$$D_1 : [(A, B, C, D); (A, D, E); (B, F); (F, G, H)]$$
$$D_2 : [(A, B, C); (D, E); (B, F); (F, G, H)]$$

Which is correct?

A. D_1 *is lossless,* D_2 *is lossy.*

B. D_1 *is lossy,* D_2 *is lossless.*

C. *Both* D_1 *and* D_2 *are lossy.*

D. *Both* D_1 *and* D_2 *are lossless.*

Exercise 189 *A relation is in 3NF but fails BCNF. This implies the existence of a non-trivial FD* $X \to A$ *such that:*

A. X *is not a superkey and* A *is a non-prime attribute.*

B. X *is a superkey and* A *is a prime attribute.*

C. X *is not a superkey and* A *is a prime attribute.*

D. X *is a proper subset of a candidate key and* A *is non-prime.*

Exercise 190 *Let* $F = \{AB \to C,\ B \to D,\ C \to A\}$ *hold on* $R(A, B, C, D)$*. If* R *is decomposed into* $R_1(B, D)$ *and* $R_2(A, B, C)$*:*

A. *Both* R_1 *and* R_2 *are in BCNF.*

B. *The decomposition is lossless and dependency preserving.*

C. *The decomposition is lossless but NOT dependency preserving.*

D. R_2 *is in 3NF but not BCNF.*

Exercise 191 *Identify the schema that is in 3NF but NOT in BCNF (underlined attributes form the primary key):*

A. $S_1(\underline{ID}, Name)$ *with* $\{ID \to Name\}$

B. $S_2(\underline{S\_ID, C\_ID}, Email)$ *with* $\{S\_ID, C\_ID \to Email,\ Email \to S\_ID\}$

C. $S_3(\underline{S\_ID, C\_ID}, Score, Grade)$ *with* $\{S\_ID, C\_ID \to Score,\ Score \to Grade\}$

D. $S_4(\underline{S\_ID, C\_ID}, Fee)$ *with* $\{S\_ID, C\_ID \to Fee,\ C\_ID \to Fee\}$

Exercise 192 *Functional dependencies on* $R(A, B, C, D, E)$*:*

$$A \to B, \quad AB \to C, \quad D \to AC, \quad D \to E$$

An irreducible equivalent (canonical cover) is:

A. $\{A \to B, A \to C, D \to A, D \to E\}$

B. $\{A \to B, B \to C, D \to A, D \to E\}$

C. $\{A \to B, A \to C, D \to B, D \to E\}$

D. $\{A \to B, D \to A, D \to E\}$

Exercise 193 *A relation* $R(V, N, S, E, T, Y, P)$ *tracks* $Volume, Number, Start, End, Title, Year, Price$*. The PK is* (V, N, S, E)*. FDs:* $\{(V, N, S, E) \to T,\ (V, N) \to Y,\ (V, N, S, E) \to P\}$*. It is decomposed into* $R_1(V, N, S, E, T, P)$ *and* $R_2(V, N, Y)$*. What is the highest normal form satisfied by the* ***new*** *design?*

A. *1NF*

B. *2NF*

C. *3NF*

D. *BCNF*

Exercise 194 *In relation* $R(A, B, C, D, E, F)$ *with* $F = \{\{A, C\} \to \{D, E\}, \{A, D, F\} \to \{B, C\}\}$*, which of the following is a trivial FD in* F^+*?*

A. $\{A, C\} \rightarrow \{D\}$

B. $\{A, C, D\} \rightarrow \{C, D\}$

C. $\{A, D, F\} \rightarrow \{B\}$

D. $\{A, D\} \rightarrow \{D, E\}$

Exercise 195 *A **prime attribute** in database design is defined as an attribute that:*

A. *Is the first attribute defined in the schema.*

B. *Is a member of at least one candidate key.*

C. *Cannot contain null values.*

D. *Determines all other attributes in the relation.*

Exercise 196 *Consider these statements:*

- ***S1:** Any relation with only two attributes must be in BCNF.*
- ***S2:** $\{X \rightarrow Y, Z \rightarrow W, W \rightarrow Y\}$ is a minimal cover for $\{X \rightarrow Y, Z \rightarrow W, X \rightarrow W, W \rightarrow Y\}$.*

Which is correct?

A. *Both S1 and S2 are TRUE*

B. *S1 is TRUE, S2 is FALSE*

C. *S1 is FALSE, S2 is TRUE*

D. *Both S1 and S2 are FALSE*

Exercise 197 *Given $R(A, B, C, D, E, F, G, H)$ and $F = \{A \rightarrow B,\ B \rightarrow CD,\ AE \rightarrow F,\ F \rightarrow G,\ G \rightarrow H\}$. The candidate key for R is:*

A. $\{A, E\}$

B. $\{A, B, E\}$

C. $\{A, E, G\}$

D. $\{A\}$

Exercise 198 *Relation $R(A, B, C, D, E, F, G, H)$ with FDs:*

$$BC \rightarrow D, \quad A \rightarrow BC, \quad B \rightarrow CEF, \quad G \rightarrow A, \quad E \rightarrow GH$$

The relation R is in:

A. *1NF but not 2NF*

B. *2NF but not 3NF*

C. *3NF but not BCNF*

D. *BCNF*

Exercise 199 *For the relation R defined in the previous exercise (Exercise 22), how many candidate keys exist?*

A. *2*

B. *3*

C. *4*

D. *5*

Exercise 200 *Which of the following statements is TRUE?*

A. *A 3NF relation is always in BCNF if it has no overlapping candidate keys.*

B. *BCNF always preserves all functional dependencies.*

C. *2NF allows transitive dependencies between non-prime attributes.*

D. *1NF relations cannot have composite keys.*

Exercise 201 *Consider $R(A, B, C, D)$ with FDs $\{A \to B,\ B \to C,\ C \to D\}$. If we decompose R into $R_1(A, B), R_2(B, C), R_3(C, D)$:*

A. *The decomposition is lossy.*

B. *The decomposition is lossless and dependency preserving.*

C. *The decomposition is lossless but NOT dependency preserving.*

D. *The decomposition is in 3NF but not BCNF.*

Exercise 202 *Relation $R(X, Y, Z, W)$ has FDs $\{X \to Y,\ Z \to W\}$. Decomposition into $R_1(X, Y)$ and $R_2(Z, W)$ is:*

A. *Lossless join.*

B. *Dependency preserving.*

C. *Both lossless and dependency preserving.*

D. *Neither.*

Exercise 203 *Relation $R(A, B, C, D, E)$ with FDs $\{A \to B,\ B \to C,\ D \to A\}$. The highest normal form is:*

A. *1NF*

B. *2NF*

C. *3NF*

D. *BCNF*

Exercise 204 *Consider the following instance of $R(A, B, C)$:*

A	B	C
1	10	100
1	10	200
2	20	300

Which FD is satisfied by this instance?

A. $A \to C$

B. $B \to C$

C. $B \to A$

D. $C \to B$

Exercise 205 *Consider $R(A, B, C, D)$ with $\{AB \to C,\ C \to D,\ D \to B\}$. Which statements are TRUE? (MSQ)*

A. *$\{A, B\}$ is a candidate key.*

B. *$\{A, C\}$ is a candidate key.*

C. $\{A, D\}$ *is a candidate key.*

D. *The relation is in 3NF.*

Exercise 206 *If a relation $R(P, Q, R)$ is decomposed into $R_1(P, Q)$ and $R_2(P, R)$, the join is lossless if:*

A. $P \to Q$

B. $Q \to R$

C. $P \to R$

D. *Either A or C*

Exercise 207 *Given $R(A, B, C)$ and the instance:*

A	B	C
x	y	z
x	y	w
p	q	r

Which of the following is a candidate key?

A. $\{A, B\}$

B. $\{A, C\}$

C. $\{B, C\}$

D. $\{A\}$

Exercise 208 *Decomposing $R(A, B, C, D, E)$ with $\{A \to BC,\ CD \to E\}$ into $R_1(A, B, C)$ and $R_2(A, D, E)$:*

A. *The join is lossless.*

B. *The join is lossy.*

C. $A \to E$ *is preserved.*

D. *Both A and C.*

Exercise 209 *In a BCNF decomposition of $R(U, V, W, X)$ with $\{U \to V,\ V \to W,\ W \to X\}$, how many relations are produced in the final result? (NAT)*

Exercise 210 *Instance of $R(A, B, C, D)$:*

A	B	C	D
1	1	1	1
2	1	1	1
1	2	1	2

Which of the following FDs is satisfied?

A. $A \to B$

B. $B \to C$

C. $C \to D$

D. $D \to B$

6.10 YouTube Links and QR Codes

Lecture	Details	YouTube Link	QR Code
36	Functional Dependency & Armstrong's Axioms (Rules of Inference) — Normalization	https://youtu.be/7pinflaeecA	
37	Finding Candidate Keys — Normalization	https://youtu.be/SYvkOUbQZ8A	
38	Equivalence of FDs & Minimal Cover	https://youtu.be/EaA23i3vGJ4	
39	Lossy vs Lossless Decomposition — Chase Algorithm Explained	https://youtu.be/3q0rfTMxif0	

40	Dependency Preserving Decomposition	https://youtu.be/FKFKO52iK6A	
41	Normal Forms — 1NF & 2NF Explained (Easy & Intuitive)	https://youtu.be/oixd_w4IGss	
42	Third Normal Form (3NF) — Removing Transitive Dependency	https://youtu.be/E0n6pxWRRDk	
43	BCNF Decomposition — Boyce-Codd Normal Form Explained	https://youtu.be/CStTBVmNOL8	
44	Multivalued Dependency (MVD) & Fourth Normal Form (4NF)	https://youtu.be/Nr1XlCvuBw4	

45	Join Dependency & Fifth Normal Form (5NF)	https://youtu.be/yQBR1mpGc7g	
46	Practice Problems on Normalization — FD, Closure, Keys, Decomposition, Normal Forms	https://youtu.be/QqIpR3hJBuc	
47	GATE PYQs on Normalization — FD, Closure, Keys, Decomposition, Normal Forms	https://youtu.be/iE93td4uU0s	

Chapter 7

Data Organization

7.1 Record Organization

Record Organization

Record organization defines how records (tuples) are physically stored inside database files on disk. The choice of record organization directly affects search cost, insertion cost, deletion cost, and range query performance.
A file is a collection of records stored in blocks. Different file organization methods optimize different types of operations such as lookup, scan, update, and join.

Why Record Organization Matters

- Determines how quickly records can be located.
- Impacts insertion and deletion performance.
- Affects disk I/O cost and block access patterns.
- Influences index design and query optimization.
- Different workloads require different organization strategies.

Basic Concepts

- A **record** is a collection of related fields stored together.
- A **file** is a collection of records.
- A **block (page)** is the unit of disk I/O.
- Record organization defines how records are arranged inside blocks and files.
- Organization is independent of logical schema but critical for performance.

7.1.1 Records and Record Types

Records and Record Types

Data in database systems is typically stored in the form of **records**. A record is a collection of related data values or data items, where each value corresponds to a particular **field** of the record.
Each field value consists of one or more bytes and represents a specific attribute of an entity. Records are used to describe entities and their attributes in a structured way.
For example, an **EMPLOYEE** record represents an employee entity. Each field in the record stores information about a specific attribute of the employee such as:

- Name
- Birth_date
- Salary
- Supervisor

Thus, a record can be viewed as a structured representation of a real-world entity.
A **record type** (or **record format**) defines the structure of records by specifying:

- The list of field names
- The data type of each field
- The size of each field

Each field has an associated **data type** that determines the set of values the field can store.

Common Data Types Used in Records

Database systems typically use standard programming data types for fields.

Data Type	Example Value	Typical Size
Integer	1024	4 bytes
Long Integer	9876543210	8 bytes
Real / Float	45.78	4 bytes
Boolean	TRUE / FALSE	1 byte
Date	2026-03-25	10 bytes
Fixed-length String	"SMITH"	k bytes
Variable-length String	"Jonathan"	number of characters

Observations

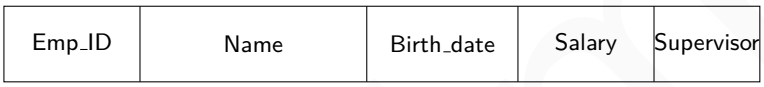
- Numeric data types have fixed storage sizes.
- Fixed-length strings occupy the same number of bytes regardless of actual value length.
- Variable-length strings require space proportional to the number of characters stored.
- Date and time fields often use specially encoded formats.

Example Record Type: EMPLOYEE

A record format defines the fields and their data types.

Field Name	Data Type	Size (bytes)
Emp_ID	Integer	4
Name	Varchar(30)	variable
Birth_date	Date	10
Salary	Float	4
Supervisor_ID	Integer	4

This table specifies the structure of the EMPLOYEE record type.



EMPLOYEE Record Structure

Binary Large Objects (BLOBs)

In many modern database applications, it is necessary to store large unstructured data items.
Examples include:

- Images
- Audio recordings
- Video streams
- Large text documents

These types of data are stored as **Binary Large Objects (BLOBs)**.
Characteristics of BLOBs:

- They store large binary data.
- They may occupy thousands or millions of bytes.
- They are usually stored outside the normal record structure.
- Records store only a pointer or reference to the BLOB location.

BLOB storage is commonly used in multimedia databases and document management systems.

Example 283: EMPLOYEE Record Instance

Consider the EMPLOYEE record type defined earlier.
An example record stored in the database may be:

Emp_ID	Name	Birth_date	Salary	Supervisor_ID
101	Alice	1995-04-12	55000	205

Interpretation:

- Employee ID is 101
- Employee name is Alice
- Birth date is April 12, 1995
- Salary is 55,000
- Supervisor ID is 205

Thus, a **record instance** stores actual data values corresponding to the defined record type.

7.1.2 Files, Fixed-Length Records, and Variable-Length Records

Files Fixed-Length Records and Variable-Length Records

A **file** is a collection or sequence of records stored on secondary storage. In most database systems, all records within a file belong to the same **record type**, meaning they share the same structure of fields.
Depending on the structure of records, files can be classified into two categories:

- **Fixed-Length Record Files**
- **Variable-Length Record Files**

Fixed-Length Records
If every record in a file occupies exactly the same number of bytes, the file is said to contain **fixed-length records**. In this case:

- Each field has a fixed size.
- Every record has identical structure.
- The starting position of each field can be calculated directly.

This simplifies record access because the system can easily compute the location of any record using:

$$\text{Record Address} = \text{Base Address} + (i - 1) \times \text{Record Size}$$

where i is the record number.

Reasons for Variable-Length Records

A file may contain **variable-length records** when records have different sizes.
This occurs due to several reasons:

- **Variable-Length Fields**

  Some fields may not have fixed size. For example, the *Name* field in an EMPLOYEE record may contain names of different lengths.

- **Repeating Fields**

  Certain fields may contain multiple values for a single record. Such fields are called **repeating fields**, and the collection of values is called a **repeating group**.

  Example: A student may have multiple phone numbers.

- **Optional Fields**

  Some fields may exist for certain records but not for others.

  Example: A *Middle_Name* field may exist only for some employees.

- **Mixed Record Types**
  A file may contain records of different record types with different sizes.
  Example: A STUDENT record followed by several GRADE_REPORT records for that student.

Fixed-Length Record Format

In a fixed-length record file, every field has a predetermined number of bytes.

Name	SSN	Salary	Job_Code	Dept	HireDate

Fixed-Length Record (Total size = 71 bytes)

Characteristics:

- All fields have fixed storage sizes.
- The position of each field is predetermined.
- Direct byte offsets can be used to locate fields.
  Example:
  - Name starts at byte 1
  - SSN starts at byte 31
  - Salary starts at byte 40

Variable-Length Record Using Separators

When records contain fields of varying length, separator characters can be used to identify field boundaries. Special characters such as $, |, or # may be used as field separators.

Smith,John | 123456789 | 50000 | Computer

Variable-Length Record Using Field Separators

In this representation:

- Field lengths vary depending on the stored value.
- Separator characters indicate where one field ends and the next begins.

Variable-Length Records with Field-Value Pairs

Another approach is to store each field as a $< field-name, field-value >$ pair.

Name = Smith, John
SSN = 123456789
Department = Computer

Record Stored as $< Field, Value >$ Pairs

This format is useful when:

- The number of possible fields is large.

- Only a few fields appear in most records.
- Many fields are optional.

A more compact implementation may store $< field-typecode, field-value >$ pairs instead of full field names.

Example 284: File Containing Records

Consider a file storing EMPLOYEE records.

File = Sequence of Records

Record 1
Record 2
Record 3
Record 4

Thus:

- A file is simply a **sequence of records**.
- Each record corresponds to one entity instance.
- Records may be either fixed-length or variable-length.

7.1.3 Record Blocking and Spanned versus Unspanned Records

Unspanned Organization

In an **unspanned organization**, a record must be stored completely inside a single block.

- Records are **not allowed** to cross block boundaries.
- If a record cannot completely fit in the remaining space of a block, that space remains unused.
- This approach simplifies record access since record locations are predictable.

Advantages

- Simple record processing
- Direct computation of record locations
- Efficient for fixed-length records

Disadvantage

- May waste disk space due to unused block space

Record 1
Record 2
Record 3
Record 4

Unspanned Organization

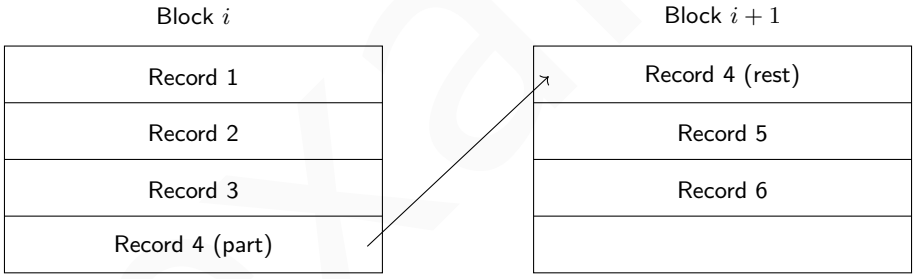
Spanned Organization

In a **spanned organization**, a record is allowed to span across multiple blocks.
If a record cannot fit completely in the remaining space of a block:

- Part of the record is stored in the current block
- The remaining part is stored in the next block

A **pointer** is used to indicate where the remainder of the record continues.
Spanned organization is necessary when:

- Record size $R > B$
- Variable-length records are large
- Storage efficiency is important



Spanned Organization

Record Blocking (unspanned)

Records stored in files must be placed inside **disk blocks** because a block is the basic unit of data transfer between disk and main memory.
Let

- $B =$ Block size (bytes)
- $R =$ Record size (bytes)

If $B \geq R$, multiple records can fit inside a single block.
The number of records stored in a block is called the **Blocking Factor (bfr)**.

$$bfr = \left\lfloor \frac{B}{R} \right\rfloor$$

Unused space in a block is

$$\text{Unused Space} = B - (bfr \times R)$$

Record Blocking (Spanned)

Records stored in files must be placed inside **disk blocks** because a block is the basic unit of data transfer between disk and main memory.
In **spanned organization**, a record is allowed to cross block boundaries. A record that does not completely fit in the remaining space of a block continues in the next block.
Let

- B = Block size (bytes)
- R = Record size (bytes)

Because records may span across blocks, the average number of records per block is

$$bfr = \frac{B}{R}$$

Unused space inside blocks is minimized because leftover space in a block can be used to store part of the next record.

Blocks Required for a File (Unspanned)

If a file contains r records and the blocking factor is bfr, the number of blocks required is

$$b = \left\lceil \frac{r}{bfr} \right\rceil$$

where $\lceil x \rceil$ denotes the **ceiling function**.

Blocks Required for a File (Spanned)

If a file contains r records and each record has size R bytes, then the total file size is $r \times R$
Let B be the block size. The number of blocks required is

$$b = \left\lceil \frac{r \times R}{B} \right\rceil$$

where $\lceil x \rceil$ denotes the **ceiling function**. This formula is used for **spanned organization** because a record can span across multiple blocks.

Example 285:

A file has block size $B = 1024$ bytes and record size $R = 100$ bytes.
Find the **blocking factor** and the **unused space per block** assuming unspanned. Calculate **blocking factor** for spanned.
Solution:
For **unspanned organization**

$$\text{Blocking Factor} = \left\lfloor \frac{B}{R} \right\rfloor$$

$$= \left\lfloor \frac{1024}{100} \right\rfloor = 10$$

Unused space per block:

$$1024 - (10 \times 100) = 24 \text{ bytes}$$

For **spanned organization**

$$\text{Blocking Factor} = \frac{B}{R} = \frac{1024}{100} = 10.24$$

Example 286:

A file contains 10,000 records. Block size $B = 4096$ bytes and record size $R = 200$ bytes.
Find the **number of blocks required** assuming unspanned organization.
Solution:
Blocking factor:

$$bfr = \left\lfloor \frac{4096}{200} \right\rfloor = 20$$

Number of blocks required:

$$\text{Blocks} = \left\lceil \frac{10000}{20} \right\rceil = 500$$

Example 287:

Block size $B = 512$ bytes and record size $R = 800$ bytes.
Determine whether **spanned** or **unspanned organization** must be used.
Solution:
In **unspanned organization**, a record must fit completely inside one block.
Since

$$R = 800 > B = 512$$

the record cannot fit into a single block.
Therefore, **unspanned organization is not possible**.
Hence, the file must use **spanned organization** where a record can span multiple blocks.

Example 288:

A file contains 12,500 records. Block size $B = 2048$ bytes and record size $R = 150$ bytes.
Find the **blocking factor** and the **number of blocks required** assuming unspanned organization.
Solution:
Blocking factor:

$$bfr = \left\lfloor \frac{2048}{150} \right\rfloor = 13$$

Number of blocks:

$$\text{Blocks} = \left\lceil \frac{12500}{13} \right\rceil$$

$$= 962$$

Example 289:

A file uses **unspanned organization**. Block size $B = 4096$ bytes and record size $R = 300$ bytes.
Find the **blocking factor**, **unused space per block**, and **total wasted space if the file has 200 blocks**.
Solution:
Blocking factor:

$$bfr = \left\lfloor \frac{4096}{300} \right\rfloor = 13$$

Space used in each block:

$$13 \times 300 = 3900$$

Unused space per block:

$$4096 - 3900 = 196 \text{ bytes}$$

Total wasted space for 200 blocks:

$$200 \times 196 = 39200 \text{ bytes}$$

Example 290:

A file uses **spanned organization**. Block size $B = 4096$ bytes and record size $R = 300$ bytes.
The file contains **200 records**.
Find the **number of blocks required** to store the file.
Solution:
Total data size:

$$200 \times 300 = 60000 \text{ bytes}$$

Number of blocks required:

$$\text{Blocks} = \left\lceil \frac{60000}{4096} \right\rceil$$

$$= 15$$

7.2 Organization of Records in Files

Organization of Records in Files

1. Heap File Organization
2. Sequential File Organization
3. Hash File Organization
4. Multitable Clustering Organization

Heap File Organization (Unordered)

In **Heap File Organization**, records are stored in the order in which they are inserted. There is **no ordering based on any search key**. New records are simply placed in the next available free space.
This organization is simple and supports very fast insert operations.
However, searching for a particular record requires scanning the file sequentially.
Search Cost
If the file occupies b disk blocks,

$$\text{Worst case block accesses} = b$$

$$\text{Average case block accesses} = \frac{b}{2}$$

Insertion Cost
Usually

$$1 \text{ block access}$$

because the record is appended to the last block.

Block 1	Block 2	Block 3
R7	R1	R8
R2	R5	R4
R9	R3	Free

Example 291:

A heap file contains $b = 120$ blocks.
Find the number of block accesses required to search a record.
Solution
Worst case

$$120 \text{ block accesses}$$

Average case

$$\frac{120}{2} = 60$$

Thus heap files are efficient for insertion but inefficient for searching.

Sequential File Organization

In **Sequential File Organization**, records are stored in **sorted order of a search key**.
Because the records are ordered, searching can be performed using **binary search**.
This organization is efficient for applications that frequently process records in sorted order.
However, insertion and deletion are expensive because the ordering must be preserved.
Search Cost
If the file has b blocks,

$$\text{Binary search cost} = \log_2 b$$

Insertion Cost
Records may need to be shifted.
Worst case

$$b \text{ block accesses}$$

Block 1	Block 2	Block 3
10	40	70
20	50	80
30	60	90

Example 292:

Suppose a sequential file contains

$$b = 1024 \text{ blocks}$$

Find the block accesses required for searching.
Solution

$$\log_2 1024 = 10$$

Therefore,

$$10 \text{ block accesses}$$

are required using binary search.

Hash File Organization

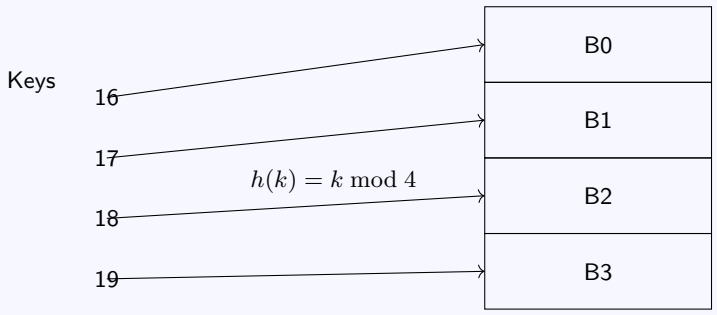
In **Hash File Organization**, records are distributed across blocks using a **hash function** applied to the search key.

$$h(k) = k \bmod m$$

where m is the number of buckets.
Hashing is very efficient for **exact-match queries**.
However it is inefficient for range queries.
Search Cost
Ideally

$$1 \text{ block access}$$

because the hash function directly identifies the block.
If collisions occur, overflow blocks may need to be accessed.



Example 293:

Given hash function

$$h(k) = k \bmod 4$$

Find the block where key $k = 22$ will be stored.
Solution

$$h(22) = 22 \bmod 4 = 2$$

Thus the record will be stored in

$$\text{Block } B2$$

Block accesses required

$$1$$

Multitable Clustering Organization

In **Multitable Clustering Organization**, records from multiple related tables are stored together in the same block. This improves the performance of queries that frequently join these tables.
For example, a **STUDENT** record and all corresponding **ENROLLMENT** records can be stored in the same block.
Advantage
Reduces the number of disk accesses required for join operations.

Clustered Block

Student(101, Ravi)
Enroll(101, DBMS)
Enroll(101, OS)

Example 294:

Suppose retrieving a student and all enrolled courses.
Without clustering

$$2 \text{ block accesses}$$

- Student table block
- Enrollment table block

With clustering

$$1 \text{ block access}$$

because related records are stored together.

7.3 Indexing

Indexing in Databases

Indexing is a technique used to improve the speed of data retrieval from a file.
Instead of scanning every record in the file, an **index structure** stores **search key values along with pointers to the blocks or records** where the data is stored.
An index works similar to the **index of a book**. Instead of reading the entire book to find a topic, we use the index to directly locate the page number.
In databases, the index allows the system to quickly locate the block containing the required record.
Basic Idea

- Data records are stored in disk blocks.
- An index stores **(search key, pointer)** pairs.
- The pointer refers to the block containing the record.

Effect on Block Access
If a file occupies b blocks:

$$\text{Without index search cost} = b$$

$$\text{With index search cost} \approx \log_2(b)$$

Thus indexing significantly reduces disk I/O.

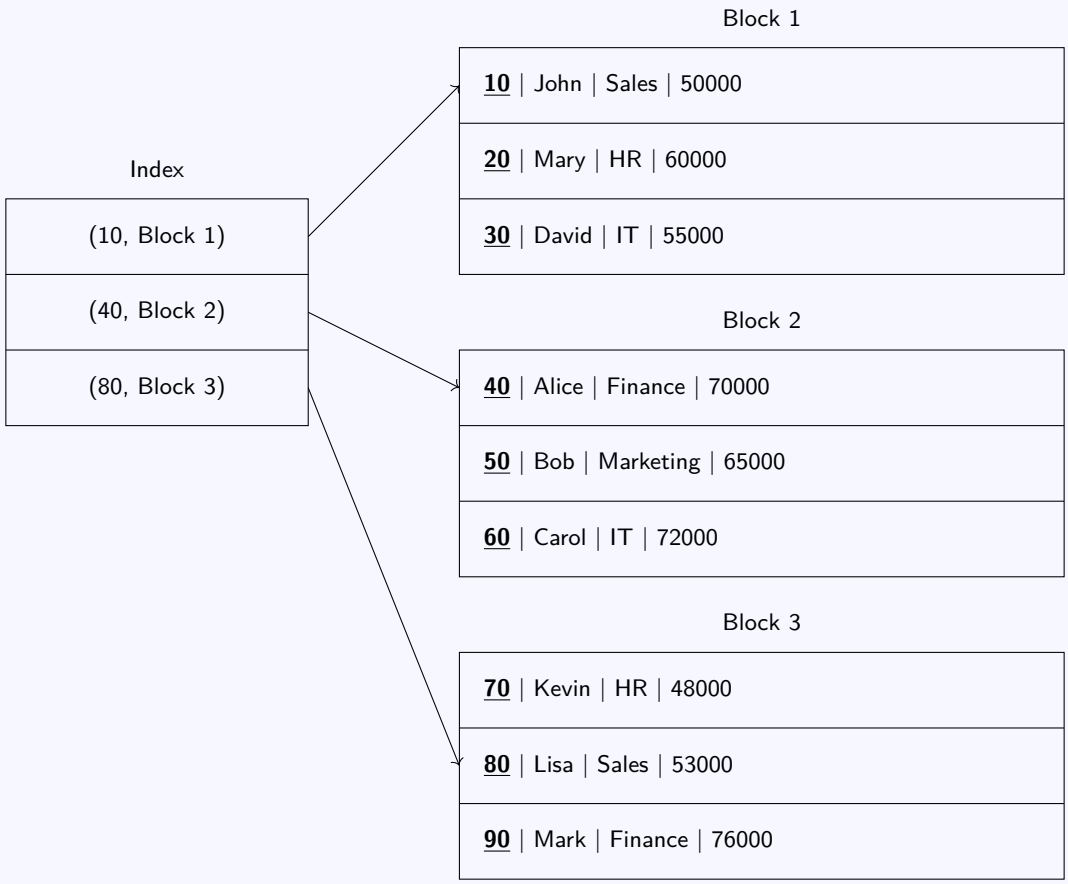
Why Indexing is Needed

Sequential search in large files is expensive because disk I/O operations are slow.
Suppose a database file has thousands of blocks. To find one record, the system may need to read many blocks.
Indexing solves this problem by providing a **fast lookup structure**.
Advantages of Indexing

- Reduces number of block accesses required to locate records.
- Improves performance of **search queries**.
- Speeds up **range queries** and **join operations**.
- Essential for large databases where sequential scanning is costly.

How Indexing Works

Consider a file containing employee records stored sequentially on disk blocks.
Each block contains multiple records.
An index file is created that stores the **search key (Employee ID)** and a **pointer to the block containing that record**.



Explanation

- The index stores the first key value of each block.
- Each index entry contains

$$(\text{Search Key}, \text{Block Pointer})$$

- When searching for a key, the system first searches the index.
- The index directly points to the correct data block.

Example 295: Effect of Indexing on Search Cost

Suppose a file contains

$$b = 1000 \text{ blocks}$$

Without Index
To find a record, the system may scan the file sequentially.

$$\text{Worst case block accesses} = 1000$$

$$\text{Average case} = 500$$

With Index
Assume the index has 100 entries.
Binary search on index:

$$\log_2(100) \approx 7$$

Then one additional access to retrieve the data block.

$$\text{Total block accesses} = 7 + 1 = 8$$

Conclusion

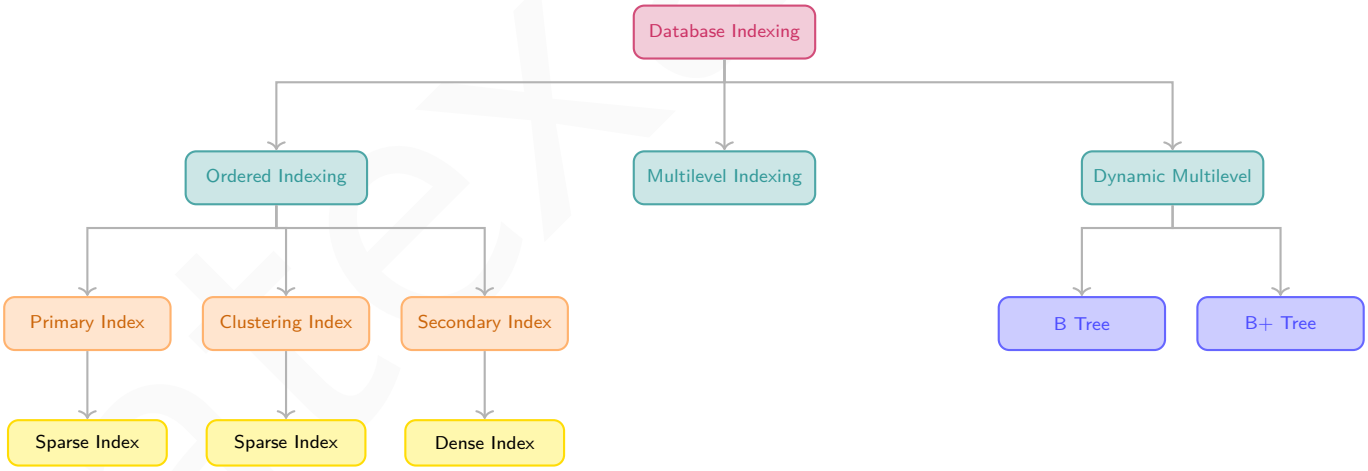
$$1000 \rightarrow 8 \text{ block accesses}$$

Thus indexing dramatically improves query performance.

Types of Indexes

Types of Indexes

1. Ordered Indexing
   - Primary Index
   - Clustering Index
   - Secondary Index
   - Dense Index
   - Sparse Index
2. Multilevel Indexing
3. Dynamic Multilevel Indexing (B Trees, B+ Trees)



7.3.1 Ordered Indexing

Ordered Indexing

Ordered indexing is a technique where index entries are stored in sorted order based on the value of the search key. Each index entry typically contains:

$$(\text{Search Key}, \text{Pointer to Record or Block})$$

Because the index is ordered, the DBMS can perform efficient searches using binary search or range scanning. Ordered indexing is commonly used in database systems for queries such as:

- Equality search (find record with a specific key)
- Range queries (e.g., salary $>$ 50000)
- Ordered retrieval (sorted results)

Block Access Cost
If the index has b_i blocks, then search requires:

$$\log_2(b_i) + 1 \text{ block accesses}$$

where:

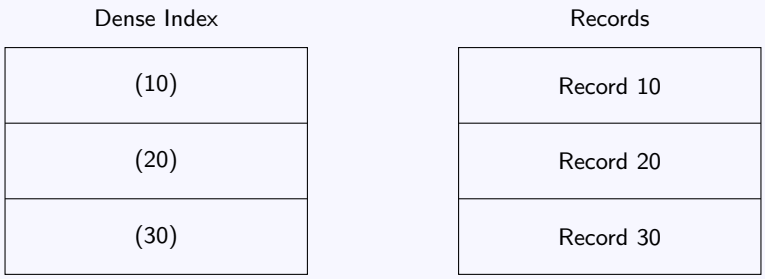
- $\log_2(b_i) =$ searching the index
- $+1 =$ accessing the data block

Dense Index

A Dense Index has an index entry for every search-key value in the data file.
Each entry points directly to the corresponding record.
Characteristics

- One index entry per record
- Faster search
- Larger index size

Example

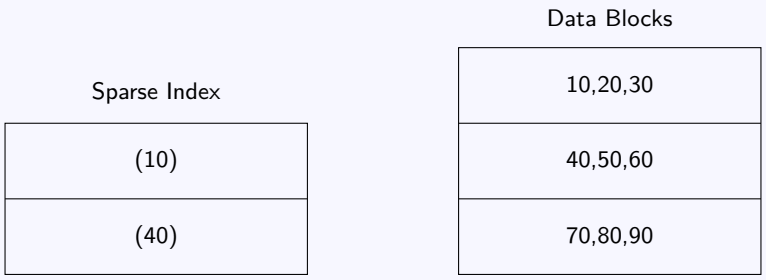


Sparse Index

A Sparse Index contains entries for only some search-key values, usually one per block.
Each entry points to the first record in the block.
Characteristics

- One index entry per block
- Smaller index
- Requires block scanning

Example



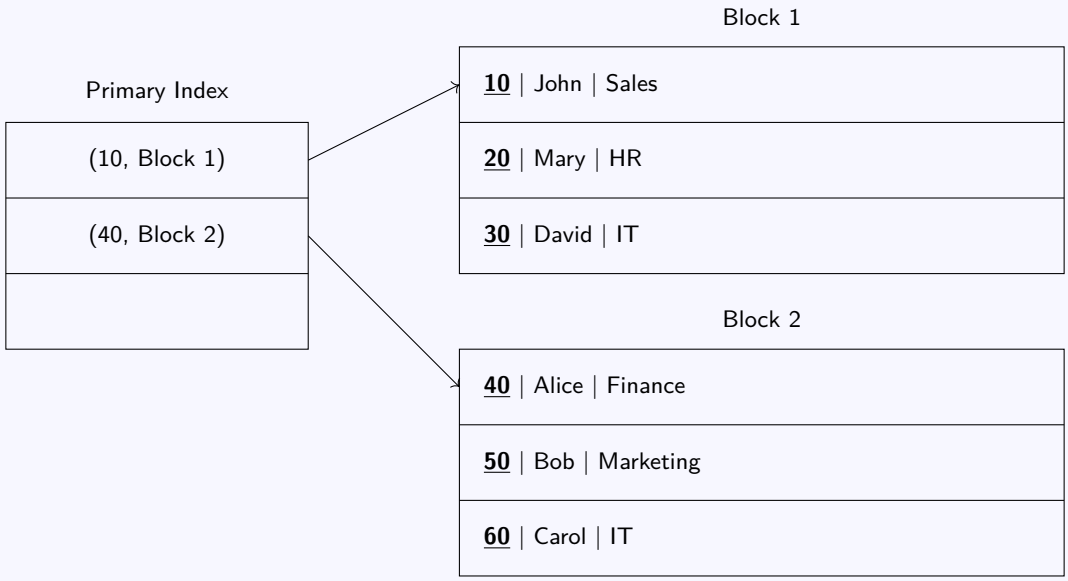
Primary Index

A Primary Index is an ordered index built on the primary key of a file where the data file itself is stored in sorted order based on the primary key.
Only one primary index can exist because the file can be physically sorted in only one way.
The primary index is usually a **sparse index**, meaning that the index stores only the first key value of each data block.

Characteristics

- Data file is sorted by primary key
- One index entry per block
- Index entry points to the block
- Usually sparse

Example
Suppose EMPLOYEE records are stored sorted by EmployeeID.



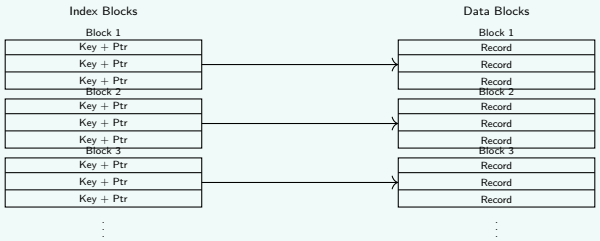
Search Example
Search EmployeeID $= 50$

1. Check index to find closest key (40)
2. Access Block 2
3. Scan block to locate record

Block access cost:

$$1 \text{ index block} + 1 \text{ data block} = 2b$$

Index and Data File Parameters (unspanned)



Index File (Index Block Access)

- r_i = number of index entries
- e = index entry size
- $bfr_i = \left\lfloor \frac{B}{e} \right\rfloor$
- $b_i = \left\lceil \frac{r_i}{bfr_i} \right\rceil$

Data File (Data Block Access)

- r = number of records
- R = record size
- $bfr = \left\lfloor \frac{B}{R} \right\rfloor$
- $b = \left\lceil \frac{r}{bfr} \right\rceil$

Total Block Accesses = Index Access + Data Block Accesses

$$\text{Index Access} = \begin{cases} 1, & b_i = 1 \\ \lceil \log_2 b_i \rceil, & b_i > 1 \end{cases}$$

Example 296: Primary Index — Block Access Calculation

Question
A file contains **10,000 employee records** stored sequentially on disk.

- Record size $R = 100$ bytes
- Block size $B = 1000$ bytes
- Index entry size $= 20$ bytes

The file is ordered on **Employee ID** and a **Primary Index** is created. Assume unspanned organization
Find:

- Blocking factor
- Number of data blocks
- Number of index entries
- Number of index blocks

- Block accesses required to search a record

Solution
Blocking factor

$$bfr = \left\lfloor \frac{B}{R} \right\rfloor$$

$$bfr = \left\lfloor \frac{1000}{100} \right\rfloor = 10$$

Number of data blocks

$$b = \left\lceil \frac{10000}{10} \right\rceil = 1000$$

Primary index contains **one entry per block**

$$\text{Index entries} = 1000$$

Index blocking factor

$$bfr_i = \left\lfloor \frac{1000}{20} \right\rfloor = 50$$

Number of index blocks

$$b_i = \left\lceil \frac{1000}{50} \right\rceil = 20$$

Binary search on index

$$\log_2 20 \approx 5$$

One more block access to fetch the record.

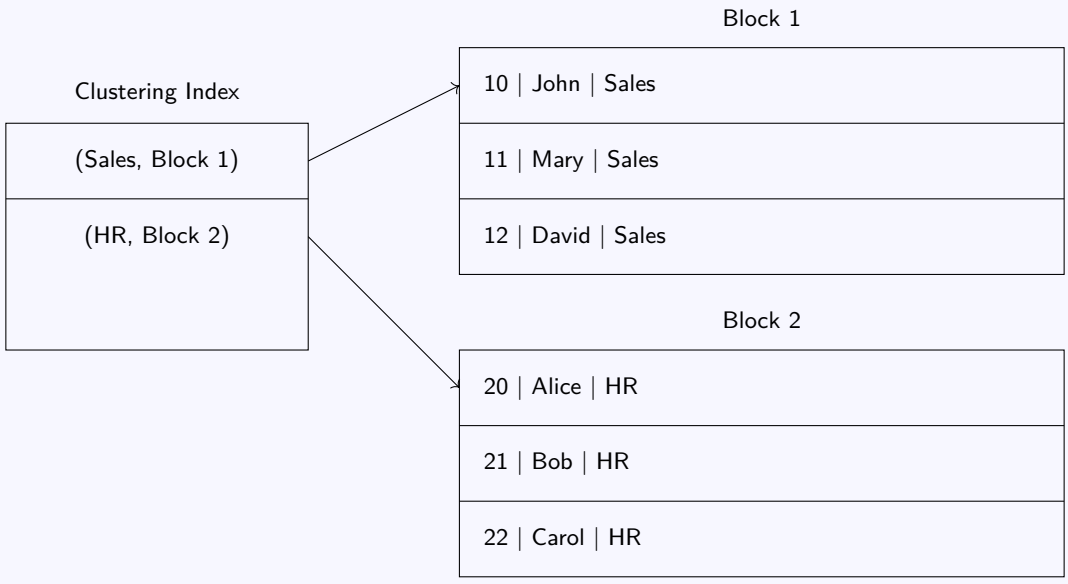
$$\textbf{Total block accesses} = 5 + 1 = 6$$

Clustering Index

A Clustering Index is built on a non-key attribute that determines the physical ordering of records.
Many records may share the same clustering attribute value.
The index entry points to the first block containing records with that value.
Characteristics

- File ordered by clustering attribute
- Attribute may have duplicate values
- One index entry per distinct attribute value

Example
Records ordered by Department.



Example 297: Clustering Index — Retrieval Cost

Question
A **STUDENT file** contains **20,000 records** ordered on attribute **Department** stored on disk using **unspanned organization**.

- Record size $R = 200$ bytes
- Block size $B = 2000$ bytes
- Index entry size $= 25$ bytes
- Number of departments $= 50$

Find the block accesses required to retrieve all students of a department.
Solution
Blocking factor

$$bfr = \left\lfloor \frac{2000}{200} \right\rfloor = 10$$

Number of data blocks

$$b = \left\lceil \frac{20000}{10} \right\rceil = 2000$$

Records per department

$$\frac{20000}{50} = 400$$

Blocks per department

$$\frac{400}{10} = 40$$

Clustering index has **one entry per department**

$$\text{Index entries} = 50$$

Index blocking factor

$$bfr_i = \left\lfloor \frac{2000}{25} \right\rfloor = 80$$

Index blocks

$$b_i = \left\lceil \frac{50}{80} \right\rceil = 1$$

Block accesses

- Index search $= 1$
- Department blocks $= 40$

Total block accesses $= 41$

Secondary Index

A Secondary Index is an index built on an attribute that is not used for physical ordering of the file. Because the file is not ordered on that attribute, the index must contain an entry for every record. Hence secondary indexes are usually dense.

Characteristics

- Data file not sorted by index attribute
- Multiple secondary indexes allowed
- Usually dense

Uses

- Enables fast search on non-key attributes
- Supports queries on attributes other than the primary key
- Useful for attributes frequently used in WHERE conditions

Limitations

- Requires extra storage for index entries
- Insert and delete operations require index updates
- May require accessing multiple data blocks when duplicates exist

Example

Index built on Salary.

Secondary Index

(50000)
(55000)
(70000)

Data Blocks

10 | John | 50000
20 | Mary | 70000
30 | David | 55000

Example 298: Secondary Index — Dense Index Access Cost

Question
A file contains **5000 employee records**.

- Record size $R = 200$ bytes
- Block size $B = 1000$ bytes
- Index entry size $= 20$ bytes

A **Secondary Index** is created on attribute **Name**.
Find the block accesses required to search for a record.
Solution
Blocking factor

$$bfr = \left\lfloor \frac{1000}{200} \right\rfloor = 5$$

Number of data blocks

$$b = \left\lceil \frac{5000}{5} \right\rceil = 1000$$

Secondary index is a **dense index**

$$\text{Index entries} = 5000$$

Index blocking factor

$$bfr_i = \left\lfloor \frac{1000}{20} \right\rfloor = 50$$

Number of index blocks

$$b_i = \left\lceil \frac{5000}{50} \right\rceil = 100$$

Binary search on index

$$\log_2 100 \approx 7$$

Access data block

$$+1$$

Total block accesses $= 8$

Example 299: Dense vs Sparse Index Comparison

Question
A file contains **12,000 records**.

- Record size $R = 120$ bytes
- Block size $B = 1200$ bytes
- Index entry size $= 24$ bytes

Find:

- Number of index entries for dense index

- Number of index entries for sparse index
- Number of index blocks for both cases

Solution
Blocking factor

$$bfr = \left\lfloor \frac{1200}{120} \right\rfloor = 10$$

Number of data blocks

$$b = \frac{12000}{10} = 1200$$

Dense index entries

$$12000$$

Sparse index entries

$$1200$$

Index blocking factor

$$bfr_i = \left\lfloor \frac{1200}{24} \right\rfloor = 50$$

Dense index blocks

$$\left\lceil \frac{12000}{50} \right\rceil = 240$$

Sparse index blocks

$$\left\lceil \frac{1200}{50} \right\rceil = 24$$

Sparse index requires significantly fewer index blocks.

Example 300: Sequential Search vs Indexed Search

Question
A file contains **40,000 records**.

- Record size $R = 200$ bytes
- Block size $B = 2000$ bytes
- Index entry size $= 40$ bytes

Compute block accesses required using:

- Sequential search
- Primary index
- Secondary index

Solution
Blocking factor

$$bfr = \frac{2000}{200} = 10$$

Number of data blocks

$$b = \frac{40000}{10} = 4000$$

Sequential search (average)

$$\frac{4000}{2} = 2000 \text{ block accesses}$$

Primary index

$$\text{Index entries} = 4000$$

Index blocking factor

$$bfr_i = \frac{2000}{40} = 50$$

Index blocks

$$b_i = \frac{4000}{50} = 80$$

Binary search

$$\log_2 80 \approx 7$$

Total

$$7 + 1 = 8 \text{ block accesses}$$

Secondary index

$$\text{Index entries} = 40000$$

Index blocks

$$b_i = \frac{40000}{50} = 800$$

Binary search

$$\log_2 800 \approx 10$$

Total

$$10 + 1 = 11 \text{ block accesses}$$

7.3.2 Multilevel Indexing

Multilevel Indexing

When a file becomes very large, even the index itself may occupy many disk blocks. Searching a large index may require several block accesses. To reduce this cost, we build an **index on the index**. This technique is called **Multilevel Indexing**.
The idea is similar to a hierarchical search structure. Instead of scanning a large index, we create multiple levels of indexes:

- **Level 1 Index** — built on the data file.
- **Level 2 Index** — built on the Level 1 index.
- Higher levels may be added if necessary.

Each higher level index contains fewer entries than the level below it. Eventually, the highest level index becomes small enough to fit into a single block.
Searching then proceeds **top-down**:

1. Search the highest level index.
2. Follow the pointer to the lower level index block.
3. Continue until the data block is located.

This greatly reduces the number of block accesses required for search operations.

Why Multilevel Indexing is Used

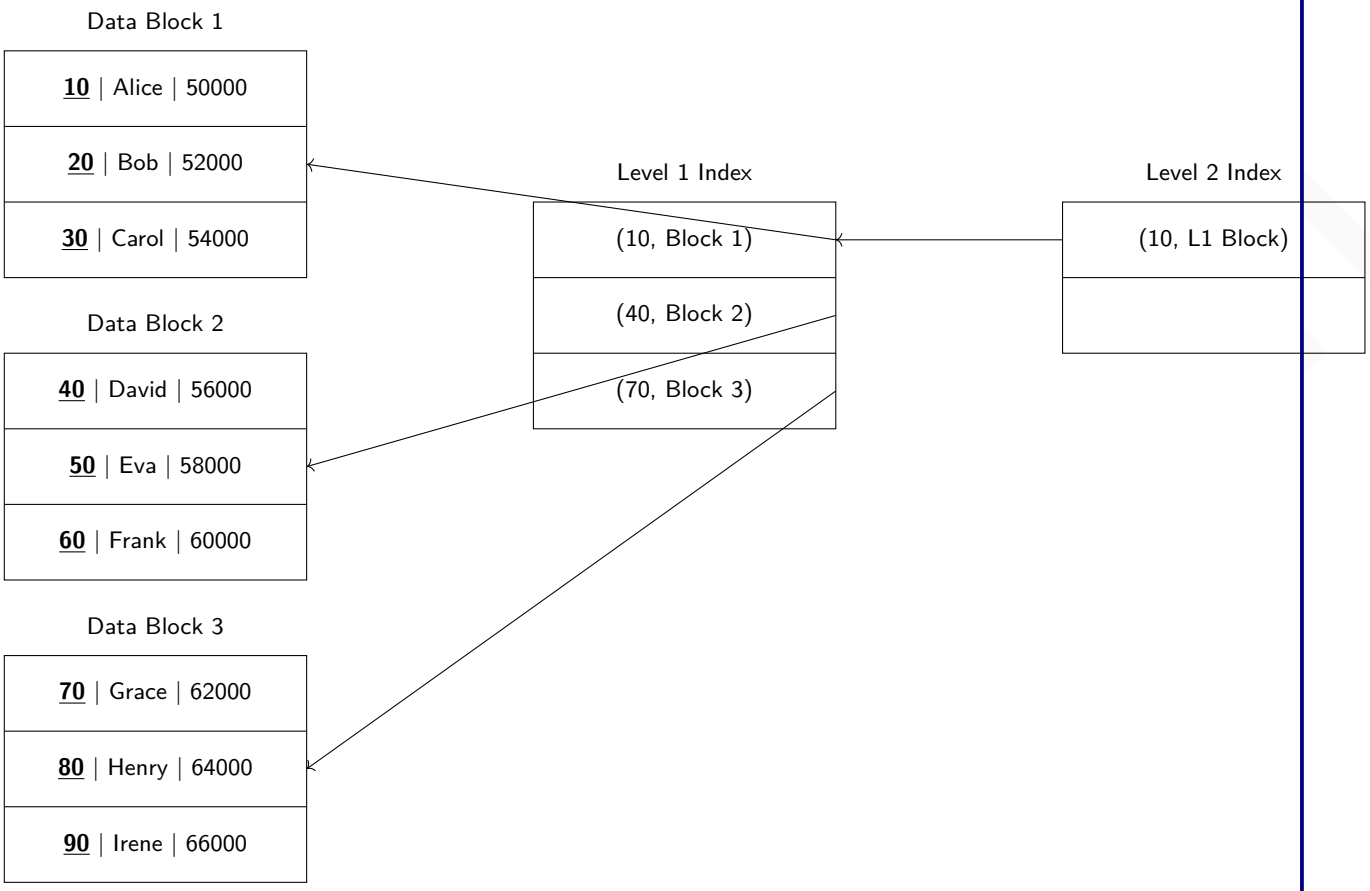
- Large indexes may span hundreds or thousands of disk blocks.
- Sequential or binary search on such large indexes would still require multiple disk accesses.
- Multilevel indexing reduces the search space at every level.
- The number of block accesses becomes proportional to the **height of the index tree** rather than the total number of records.

Example 301: Multilevel Index Structure Example

Suppose a file contains employee records ordered by **Employee ID**.

- Block size $= 3$ records per block
- Total records $= 9$

The records are stored in three data blocks.



Search Example
Suppose we want to find record with ID $= 50$.

1. Access Level 2 index block.
2. Follow pointer to Level 1 index.
3. Level 1 index indicates **Block 2**.
4. Access Data Block 2 and retrieve record.

Block Accesses

- Level 2 index $= 1$ block access
- Level 1 index $= 1$ block access
- Data block $= 1$ block access

$$\textbf{Total block accesses } = 3$$

Without indexing, sequential search may require scanning many blocks.

7.3.3 Dynamic Multilevel Indexing

Dynamic Multilevel Indexing

In large databases, the number of records continuously changes due to insertions and deletions. If we use static multilevel indexing, the index structure may become inefficient over time because block overflows and underflows occur.
To address this problem, **Dynamic Multilevel Indexing** structures are used. These indexing structures automatically reorganize themselves whenever records are inserted or deleted. The structure grows or shrinks dynamically while maintaining balanced search paths.
The most widely used dynamic multilevel index structures are:

- **B-Tree**
- **B+ Tree**

These structures ensure that:

- The index remains balanced.
- All search paths from root to leaf have the same length.
- Insertions and deletions do not degrade search performance.

Dynamic multilevel indexing is widely used in modern database systems because it supports efficient search, insertion, and deletion operations even when the database grows very large.

Properties of Dynamic Multilevel Index Structures

- The index is organized as a **balanced tree**.
- Each node corresponds to a **disk block**.
- All records are accessed by traversing the tree from the **root node to leaf nodes**.
- Insertions may cause **node splitting** when a node becomes full.
- Deletions may cause **node merging** when a node becomes too empty.
- The height of the tree remains small, ensuring fast search.

7.4 B Trees: Structure, Properties and Formulas

7.4.1 Structure

Structure of a B-Tree

A **B-Tree** is a balanced multiway search tree used in database indexing where each node corresponds to a **disk block**.
Let

$$n = \text{order of the B-tree}$$

Each node of the B-tree contains

- Search keys K_i
- Record pointers Pr_i (pointer to the record in the data file)
- Block pointers Pb_i (pointer to child blocks)

The keys in every node are stored in **sorted order**.

General Structure of a B-Tree Node

For a B-tree of order n,

$$\text{Maximum number of keys} = n - 1$$

$$\text{Maximum number of block pointers} = n$$

Thus a node may contain

$$Pb_1,\ (K_1, Pr_1),\ Pb_2,\ (K_2, Pr_2),\ Pb_3,\ \ldots,\ (K_{n-1}, Pr_{n-1}),\ Pb_n$$

B-Tree Node Diagram

Pb_1	K_1 Pr_1	Pb_2	K_2 Pr_2	Pb_3	K_3 Pr_3	Pb_4

B-tree Node Structure

Meaning of Pointers

- Pr_i
  Points to the **actual record** in the data file whose search key is K_i.
- Pb_i
  Points to a **child block** (subtree).
- Each block pointer leads to another disk block containing more keys.

Since each node is stored in a disk block, following a pointer means **one block access**.

Ordering Property of Keys

The keys inside a B-tree node satisfy the following ordering condition

$$K_1 < K_2 < K_3 < \cdots < K_{n-1}$$

The block pointers divide the key space into ranges.

$$Pb_1 \rightarrow \text{keys } < K_1$$

$$Pb_1 \rightarrow K_1 < \text{keys} < K_2$$

$$Pb_2 \rightarrow K_2 < \text{keys} < K_3$$

$$Pb_n \rightarrow \text{keys} > K_{n-1}$$

Thus each pointer represents a **range of search keys**.

B-Tree of Order n

- All **leaf nodes appear at the same level**.
- Keys in each node are stored in **sorted order**.
- Maximum children per node $= n$ Maximum keys $= n - 1$.
- Every **non-root node** has at least $\lceil n/2 \rceil$ children.
- Therefore every non-root node has at least $\lceil n/2 \rceil - 1$ keys.
- A node with k keys always has $k + 1$ **children**.

Block Size Constraint for a B-Tree Node

A B-tree node is designed so that the entire node fits inside a single disk block.
Let

$$n = \text{order of the B-tree}$$

Each B-tree node contains

$$(n-1) \text{ keys}$$

$$(n-1) \text{ record pointers}$$

$$n \text{ block pointers}$$

Let

- $K =$ size of a key
- $Pr =$ size of a record pointer
- $Pb =$ size of a block pointer
- $B =$ block size

Total space required for one B-tree node is

$$(n-1)K + (n-1)Pr + nPb$$

Since a node must fit inside a disk block of size B,

$$\boxed{(n-1)[K + Pr] + nPb \leq B}$$

This constraint determines the maximum possible order n of the B-tree.

7.4.2 Insertion

Insertion in B-Tree

Insertion in a B-tree always happens at the **leaf level**. The tree automatically maintains balance by splitting nodes when they become full.
Goal: Maintain B-tree properties

- All leaves remain at the same level.
- Keys in every node remain sorted.
- Number of keys in each node stays within allowed limits.

Basic Procedure

1. **Search the correct leaf node**

   Starting from the root, compare the key to be inserted with the keys in the node and follow the appropriate pointer until a leaf node is reached.

2. **Insert the key in sorted order**

   If the leaf node has space, simply insert the new key in the correct sorted position.

3. **Handle Overflow**

   If the node becomes full after insertion:

   - Split the node into two nodes.
   - Move the middle key to the parent node.
   - Left node contains smaller keys.
   - Right node contains larger keys.

4. **Propagate Split Upward**

   If the parent node also becomes full after receiving the promoted key:

   - Split the parent node as well.
   - Continue this process upward.

5. **Root Split**

   If the root becomes full:

   - Create a new root.
   - Middle key becomes the new root.
   - Tree height increases by 1.

Key Idea
Insertion never breaks tree balance. Whenever overflow occurs, nodes split and the middle key moves upward.

Example 302: B-Tree Insertion Step by Step

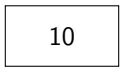
Assume a B-tree of order $n = 3$.

- Maximum keys in a node $= n - 1 = 2$
- Maximum children $= n = 3$

Insert the keys in order:

$$10,\ 20,\ 5,\ 6,\ 12,\ 30$$

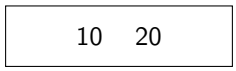
Step 1: Insert 10



The tree contains only the root.

—

Step 2: Insert 20



The root still has space.

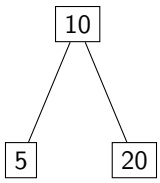
—

Step 3: Insert 5

After insertion:

$$5 \quad 10 \quad 20$$

Overflow occurs because the node can store only 2 keys.
Split the node.
Middle key 10 moves to the root.



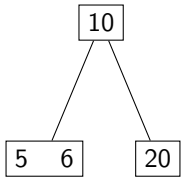
—

Step 4: Insert 6

Search path:

$$6 < 10$$

Insert into left child.



No overflow occurs.

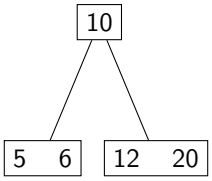
—

Step 5: Insert 12

Search path:

$$12 > 10$$

Insert into right child.



Still no overflow.

—

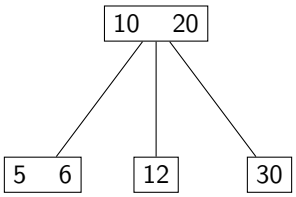
Step 6: Insert 30

Insert into right child.

$$12 \quad 20 \quad 30$$

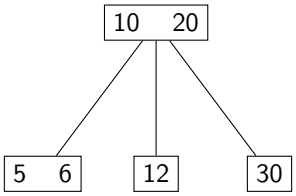
Overflow occurs.

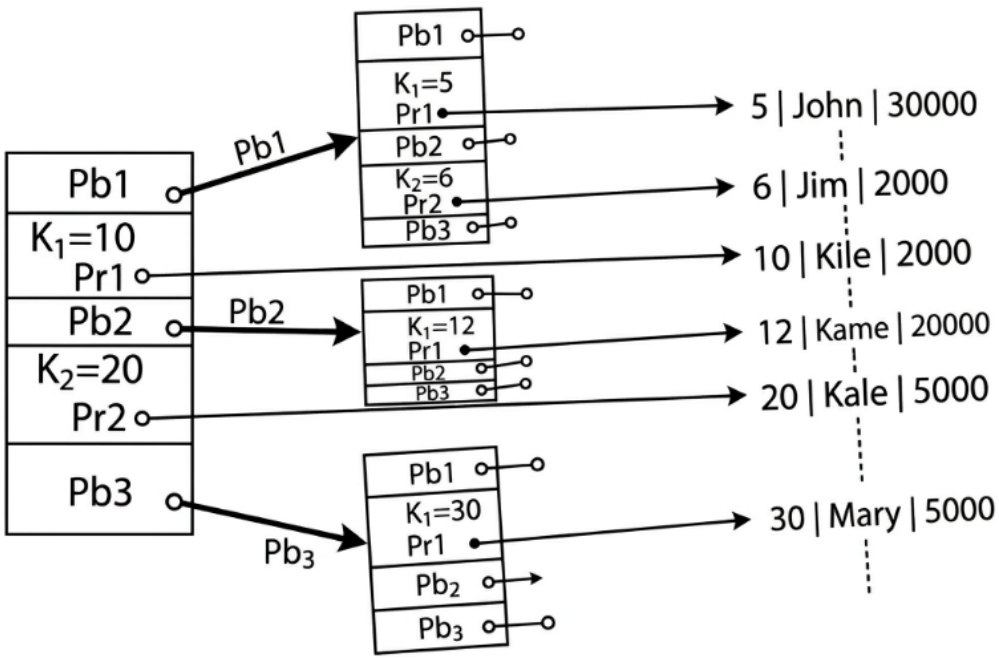
Split node and promote middle key 20.



—

Final B-tree





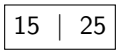
Example 303: Insertion

Insertion of 15, 25, 5, 10, 20, 30 in B-Tree of order $n = 3$ Maximum keys per node $= n - 1 = 2$

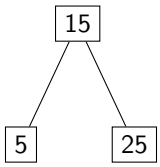
Insert 15

15

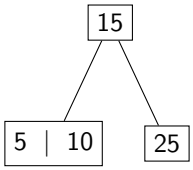
Insert 25



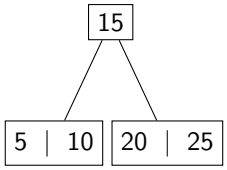
Insert 5
Overflow occurs.
Split the node and promote middle key.



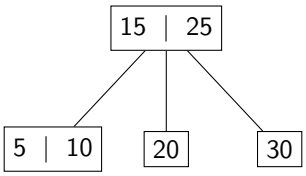
Insert 10
10 goes to left child.



Insert 20
20 goes to right child.



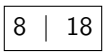
Insert 30
Overflow occurs in right child.
Split node and promote 25.



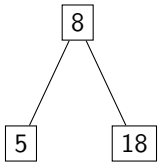
Example 304: Insertion

Insertion of 8, 18, 5, 15, 17, 25, 40 in B-Tree of order $n = 3$ Maximum keys per node $= 2$
Insert 8

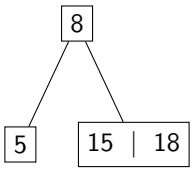
Insert 18



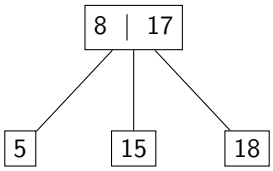
Insert 5
Overflow occurs.
Split and promote 8.



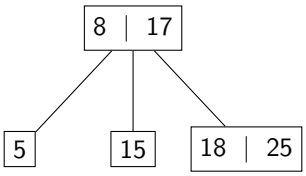
Insert 15
Insert in right child.



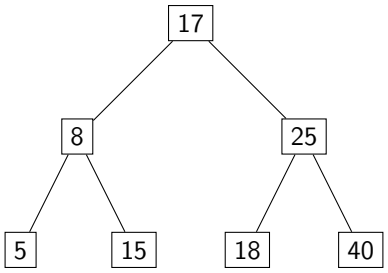
Insert 17
Overflow occurs in right child.
Split and promote 17.



Insert 25
Insert in right child.



Insert 40
Overflow occurs in right child. Split and promote 25. Root (8 — 17 — 25) also overflows. Split and promote 17.



Example 305: Finding Order of B-Tree Node

A B-tree node must fit in one disk block.
Given:

- Block size $B = 1024$ bytes
- Key size $K = 12$ bytes
- Record pointer size $Pr = 8$ bytes

- Block pointer size $Pb = 6$ bytes

Find the maximum order n of the B-tree.
Solution
B-tree node size condition

$$(n-1)K + (n-1)Pr + nPb \leq B$$

Substitute values

$$(n-1)(12) + (n-1)(8) + 6n \leq 1024$$

$$20(n-1) + 6n \leq 1024$$

$$20n - 20 + 6n \leq 1024$$

$$26n \leq 1044$$

$$n \leq 40.15$$

$$\boxed{n = 40}$$

Hence the maximum order of the B-tree is 40.

Example 306: Maximum Keys in a B-Tree Node

A B-tree is used to index employee records.
Given

- Block size $B = 2048$ bytes
- Key size $K = 16$ bytes
- Record pointer size $Pr = 8$ bytes
- Block pointer size $Pb = 8$ bytes

Find

- Maximum order n
- Maximum number of keys in a node

Solution
Node size condition

$$(n-1)K + (n-1)Pr + nPb \leq B$$

Substitute values

$$(n-1)(16) + (n-1)(8) + 8n \leq 2048$$

$$24(n-1) + 8n \leq 2048$$

$$24n - 24 + 8n \leq 2048$$

$$32n \leq 2072$$

$$n \leq 64.75$$

$$n = 64$$

Maximum keys in node

$$n - 1 = 63$$

$$\boxed{\text{Maximum keys per node} = 63}$$

Example 307: Checking Whether Node Fits in Block

Consider a B-tree node storing

- 15 keys
- 15 record pointers
- 16 block pointers

Given

- Key size $K = 10$ bytes
- Record pointer $Pr = 6$ bytes
- Block pointer $Pb = 8$ bytes
- Block size $B = 512$ bytes

Check whether the node fits in a block.
Solution
Total node size

$$15K + 15Pr + 16Pb$$

Substitute values

$$15(10) + 15(6) + 16(8)$$

$$150 + 90 + 128$$

$$= 368 \text{ bytes}$$

Since

$$368 < 512$$

$$\boxed{\text{Node fits within one block}}$$

Example 308: Designing B-Tree Node Capacity

A B-tree index is created on a large database.
Given

- Block size $B = 4096$ bytes
- Key size $K = 20$ bytes
- Record pointer $Pr = 8$ bytes
- Block pointer $Pb = 8$ bytes

Find the order n of the B-tree.
Solution
Node size condition

$$(n-1)K + (n-1)Pr + nPb \leq B$$

Substitute values

$$(n-1)(20) + (n-1)(8) + 8n \leq 4096$$

$$28(n-1) + 8n \leq 4096$$

$$28n - 28 + 8n \leq 4096$$

$$36n \leq 4124$$

$$n \leq 114.55$$

$$\boxed{n = 114}$$

Hence the B-tree node can have at most 114 child pointers.
Maximum keys

$$n - 1 = 113$$

7.5 B+ Trees: Structure, Properties and Formulas

7.5.1 Structure of B+ Trees

Structure of a B+ Tree

A **B+ Tree** is a variation of the B-Tree where **internal nodes** only store keys and child pointers, while all **actual data (record pointers)** are stored exclusively in the **leaf nodes**.
Key Differences from B-Trees:

- Internal nodes do **not** contain record pointers.
- Leaf nodes are linked sequentially using a **next-leaf pointer (Pnext)** to support efficient range queries.
- All search keys are present in the leaf nodes; some are duplicated in internal nodes for routing.

1. Internal Node Structure

For an internal node of order n, it contains only search keys and block pointers.

$$\text{Max Keys} = n - 1, \quad \text{Max Block Pointers} = n$$

Pb_1	K_1	Pb_2	K_2	Pb_3

Internal Node (Routing only)

Constraint: $(n \cdot Pb) + (n-1)K \leq B$

2. Leaf Node Structure

The leaf node stores the actual record pointers Pr_i and a **Next-Leaf Pointer** (P_{next}) to allow sequential access. Let m be the order of the leaf node.

K_1, Pr_1	K_2, Pr_2	K_3, Pr_3	P_{next}

Leaf Node (Data level)

Constraint: $m(K + Pr) + P_{next} \leq B$

Ordering and Search Property

The search keys in a B+ Tree follow a strict order:

$$K_1 < K_2 < K_3 < \cdots < K_{n-1}$$

- Pb_1 points to a subtree where all keys $K < K_1$.
- Pb_i points to a subtree where $K_{i-1} \leq K < K_i$.
- Pb_n points to a subtree where keys $K \geq K_{n-1}$.

Because internal nodes are smaller (no Pr pointers), the **fan-out** (n) is much higher than in B-Trees, leading to a shallower tree and fewer disk I/Os.

Feature	**B-Tree**	**B+ Tree**
Record Pointers	In every node	Only in Leaf nodes
Search Speed	Varies (can find in Root)	Constant (always reach Leaf)
Range Queries	Slow (requires traversal)	Very Fast (via P_{next})
Node Fill	More space for pointers	More space for keys

7.5.2 Insertion in B+ Trees

Insertion in B+ Tree

Insertion in a B+ tree always starts at the **leaf level**. The tree maintains balance by splitting nodes, but unlike B-Trees, the data storage logic differs between Leaf and Internal nodes.

Basic Procedure

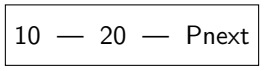
1. **Search the correct leaf node** Follow the pointers from the root to the leaf where the key should reside.
2. **Insert the key in the leaf** If the leaf has space, insert the key and record pointer in sorted order.
3. **Handle Leaf Overflow (Copy-Up Rule)** If the leaf becomes full (n keys):
   - Split leaf into L and R; keep $\lfloor n/2 \rfloor$ keys in L, move $\lceil n/2 \rceil$ keys to R.
   - **Copy** smallest key of R up to the parent.
   - Link $L \rightarrow R$ via P_{next}.
4. **Handle Internal Overflow (Move-Up Rule)** If an internal node becomes full:
   - Split the node.
   - **Move** the middle key to the parent (standard B-tree logic).
   - The middle key **is not** kept in the original node.

Key Idea: Every key must exist in the leaf level. Internal nodes only act as a "roadmap."

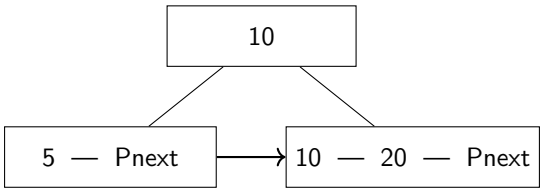
Example 309: B+ Tree Insertion Step by Step

Assume a B+ tree of order $n = 3$ Maximum keys per node $= 2$
Insert the keys: $10, 20, 5, 15, 30$

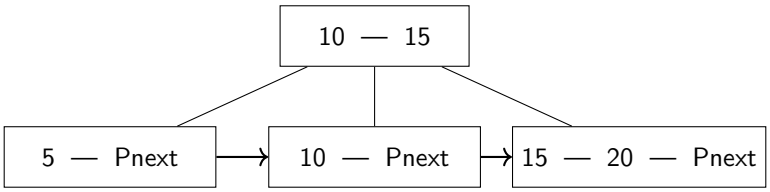
Step 1: Insert 10, 20



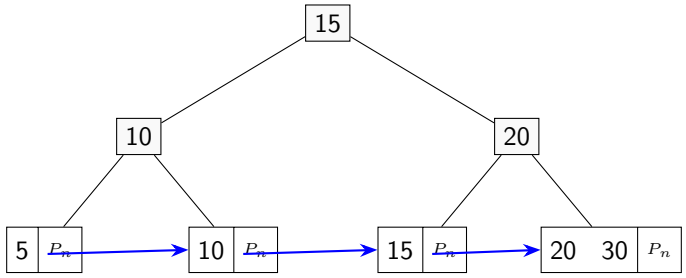
Step 2: Insert 5 (Leaf split)



Step 3: Insert 15 (Leaf split)



Step 4: Insert 30
Leaf split occurs and parent overflows $\rightarrow$ new root created.



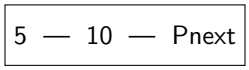
Example 310: Step-by-Step B+ Tree Insertion: 6 Elements

Assume a B+ tree of order $n = 3$.

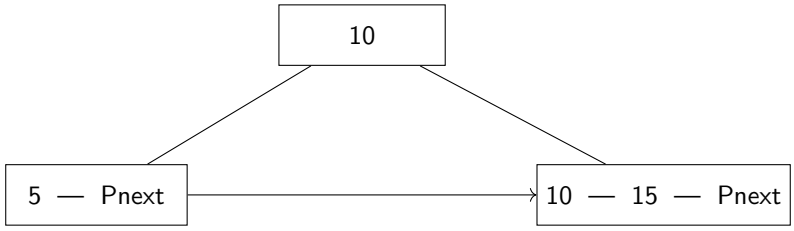
- Internal nodes: Max keys $= 2$
- Leaf nodes: Max keys $= 2$

Insert keys: $5, 10, 15, 20, 25, 30$

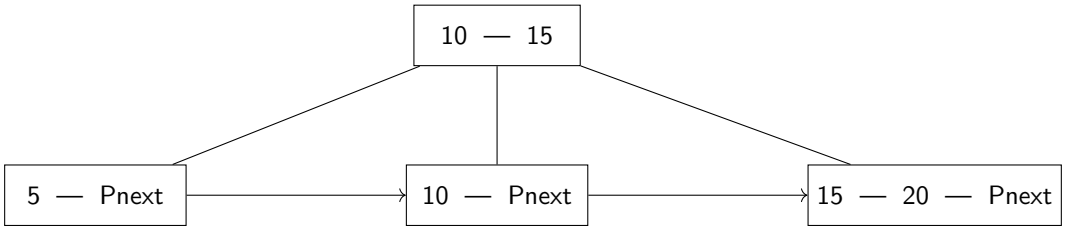
Step 1: Insert 5, 10



Step 2: Insert 15 (Leaf Split)

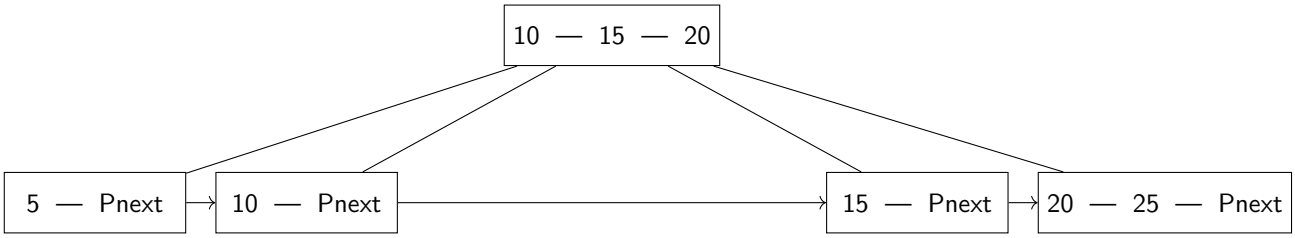


Step 3: Insert 20

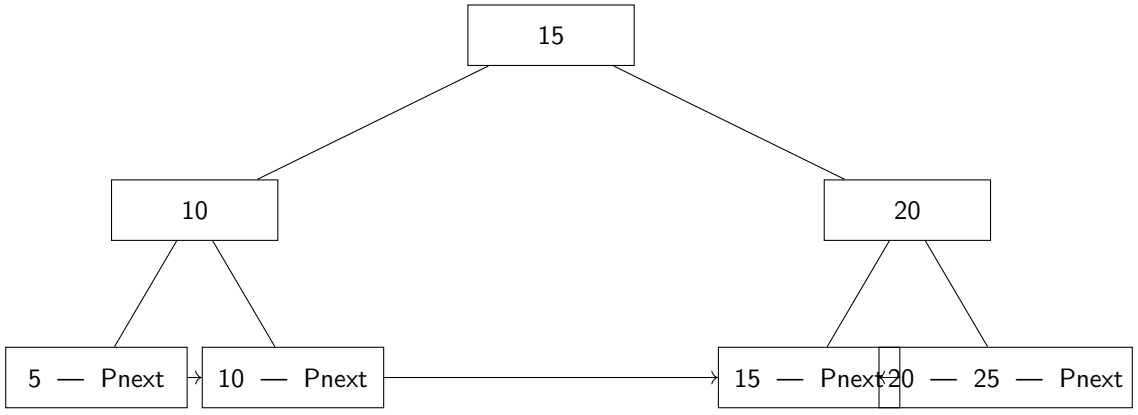


Step 4: Insert 25

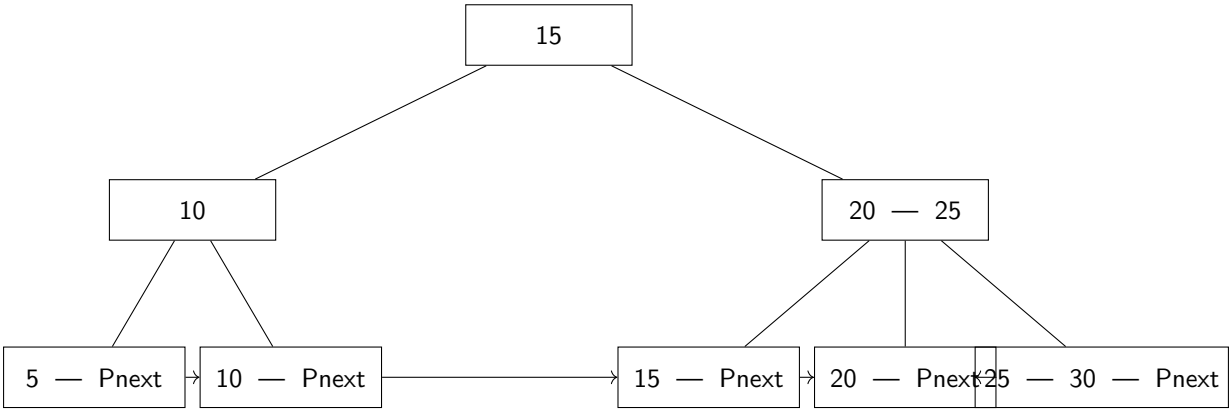
Leaf split occurs and parent overflows.



Internal node splits. Key 15 moves to new root.



Step 5: Insert 30



Final Leaf Sequence

$$5 \rightarrow 10 \rightarrow 15 \rightarrow 20 \rightarrow 25 \rightarrow 30$$

B+ Tree Block Size Constraints

A B+ tree separates navigation from data storage. Because of this, the **order n of an internal node** and the **order m of a leaf node** are calculated using different formulas.

1. Internal Node Constraint (Order n) An internal node contains n child pointers (P_i) and $n-1$ search keys (K).

$$\boxed{(n \cdot P_i) + (n-1)K \leq B}$$

2. Leaf Node Constraint (Order m) A leaf node contains m search keys (K), m record pointers (P_l), and one pointer to the next leaf block (P_{next}).

$$\boxed{m(K + P_l) + P_{next} \leq B}$$

Where:

- B = Block Size
- K = Search Key size
- P_i = Internal Child Pointer (Block Pointer)
- P_l = Leaf Record Pointer (Data Pointer)
- P_{next} = Next-leaf pointer (Sibling Pointer)

Example 311: Order Calculation for B+ Tree

Given the following parameters:

- Block Size $B = 4096$ bytes
- Search Key $K = 16$ bytes
- Internal Block Pointer $P_i = 8$ bytes
- Leaf Record Pointer $P_l = 10$ bytes
- Next-leaf Pointer $P_{next} = 8$ bytes

1. Max Order of Internal Node (n) Condition: $(n \cdot P_i) + (n-1)K \leq B$

$$(n \cdot 8) + (n-1)16 \leq 4096$$

$$8n + 16n - 16 \leq 4096$$

$$24n \leq 4112 \implies n \leq 171.33$$

$$\boxed{n = 171}$$

2. Max Order of Leaf Node (m) Condition: $m(K + P_l) + P_{next} \leq B$

$$m(16 + 10) + 8 \leq 4096$$

$$26m \leq 4088 \implies m \leq 157.23$$

$$\boxed{m = 157}$$

Example 312: Comparing B-Tree and B+ Tree Fan-out

Determine the "Fan-out" (number of children) for a B-Tree vs. a B+ Tree.
Parameters: $B = 512$, $K = 8$, $P_i = 8$, $P_l = 8$, $P_{next} = 8$.
For B-Tree: Formula: $(n-1)K + (n-1)P_l + nP_i \leq B$

$$(n-1)8 + (n-1)8 + 8n \leq 512$$

$$8n - 8 + 8n - 8 + 8n \leq 512$$

$$24n \leq 528 \implies \boxed{n = 22}$$

For B+ Tree (Internal Order n): Formula: $(n-1)K + nP_i \leq B$

$$(n-1)8 + 8n \leq 512$$

$$8n - 8 + 8n \leq 512 \implies 16n \leq 520$$

$$n \leq 32.5 \implies \boxed{n = 32}$$

Conclusion: In the same block size, a B+ Tree supports more pointers (32 vs 22), making the tree flatter and reducing disk accesses.

Example 313: Solving for Minimum Block Size

A system requires a B+ Tree with an internal order of $n = 100$. The key size $K = 20$ bytes and the internal pointer $P_i = 10$ bytes. Find the **minimum block size** B required.
Solution: Using the internal node constraint:

$$(n \cdot P_i) + (n-1)K \leq B$$

$$(100 \cdot 10) + (99 \cdot 20) \leq B$$

$$1000 + 1980 \leq B$$

$$\boxed{B \geq 2980 \text{ bytes}}$$

If the question implies standard block sizes (powers of 2), the minimum required would be **4096 bytes**.

Property	**B-Tree**	**B+ Tree**
Data Location	Found in both internal and leaf nodes.	**Exclusively** in leaf nodes.
Search Key	Not duplicated.	Duplicated (internal nodes store copies).
Range Query	Requires vertical traversals (slow).	Linear scan via P_{next} (very fast).
Internal Node	Heavier (contains Pr).	Lighter (holds more keys/pointers).

7.6 Problems

Problem 232 (NAT) *A file contains 30,000 fixed-length records of 400 bytes each. The disk block size is 2048 bytes (unspanned organization). What is the **blocking factor** (number of records per block)?*

Problem 233 (NAT) *A file contains 30,000 fixed-length records of 400 bytes each. The disk block size is 2048 bytes (unspanned organization). How many blocks does the file occupy?*

Problem 234 (MCQ) *A disk system has block size 1024 bytes. A file has 2048 records of 128 bytes each stored as a **heap file** (unspanned). What is the **maximum** number of block accesses to find a record with a specific key using linear search?*

A. 128
B. 256
C. 512
D. 1024

Problem 235 (MCQ) *Which of the following statements about **spanned** organization is correct?*

A. A record can never cross block boundaries.
B. Internal fragmentation is higher than in unspanned.
C. A record may be split across two consecutive blocks.
D. Blocking factor is always less than in unspanned organization.

Problem 236 (MSQ) *Which of the following are **advantages** of unspanned organization over spanned organization?*

A. Simpler record access logic.
B. No wasted space at end of each block.
C. No record ever needs to be read from two blocks.
D. Higher storage utilization for large records.

Problem 237 (MCQ) *Which of the following scenarios **necessarily** results in a **dense** index?*

A. A primary index on a sorted file.
B. A secondary index on a candidate key.
C. A clustering index on a non-key ordering attribute.
D. A multilevel index where the first level is sparse.

Problem 238 (MSQ) *Which of the following are true about a **sparse** index?*

A. One index entry per data block.
B. Can only be built on a sorted (ordered) file.
C. Always has fewer index entries than a dense index on the same file.
D. Supports equality search directly without accessing the data file.

Problem 239 (MCQ) *A relation $R(A, B, C)$ is stored as a sorted file on attribute A. A clustering index exists on A and a secondary index exists on B. For which of the following queries is the clustering index **not** more efficient than the secondary index?*

A. `SELECT * FROM R WHERE A > 10 AND A < 50;`
B. `SELECT * FROM R WHERE B = 100;`
C. `SELECT * FROM R WHERE A = 20;`
D. `SELECT * FROM R ORDER BY A;`

Problem 240 (NAT) *Consider a database with 30,000 records of 400 bytes each, stored as an ordered file. Block size $= 2048$ bytes (unspanned). A **sparse primary index** is built where each index entry (key + block pointer) is 30 bytes (unspanned). What is the maximum number of block accesses to retrieve a record using **binary search on the index** followed by one data block access?*

Problem 241 (MSQ) *Which of the following correctly describe a **secondary index**? (Select all that apply.)*

A. It is always built on a non-ordering field.
B. It is always a dense index.
C. It can cause multiple disk accesses even for a single record retrieval.
D. Records in the data file are physically sorted on the secondary index key.

Problem 242 (MCQ) *A database uses a **dense secondary index** on a non-key field. The data file has 10,000 blocks and the index has 200 blocks. A range query returns 1% of blocks scattered randomly in the data file. What is the **total** number of block accesses in worst case (index + data)?*

A. 200 B. 300 C. 1200 D. 10,200

Problem 243 (NAT) *Consider a student database with 1,00,000 records. A secondary index is built on attribute `Age` (50 distinct values). Each index entry contains an `Age` value (4 bytes) and a pointer to a bucket block (4 bytes). Block size = 512 bytes (unspanned). How many blocks are required for the **first level** of the index?*

Problem 244 (MCQ) *In a **multilevel index**, the first level (innermost) is a sparse primary index with 1000 entries. Each index block holds 50 entries. How many levels are needed until the index fits in **one single root block**?*

A. 1 B. 2 C. 3 D. 4

Problem 245 (NAT) *A file has 1000 blocks. A primary index is built where each index block holds 50 entries. Level 1 of the index has 20 blocks. How many blocks does **Level 2** have?*

Problem 246 (NAT) *A file has 1,00,000 records of 100 bytes each. Block size = 4 KB (unspanned). The file is sorted on the primary key. A multilevel sparse index is built where each index entry is 20 bytes (unspanned). How many **block accesses** are required to reach the data block (including all index levels)?*

Problem 247 (MSQ) *Which of the following queries benefit from a **clustering index**?*

A. Point query on the indexed attribute.
B. Range query on the indexed attribute.
C. Equality query on a non-indexed attribute.
D. `ORDER BY` on the indexed attribute.

Problem 248 (MCQ) *For a **B-tree** of order n (max children), what is the **minimum** number of keys in a non-root internal node?*

A. $\lfloor n/2 \rfloor$ B. $\lceil n/2 \rceil - 1$ C. $\lceil n/2 \rceil$ D. $\lfloor n/2 \rfloor - 1$

Problem 249 (NAT) *A B-tree of order 5 (max children) has 3 levels (root = level 1). What is the **maximum** number of keys that can be stored?*

Problem 250 (MCQ) *A B-tree is used as an index for a large database table and has **four levels** including the root. If a new key is inserted and causes splits to propagate, what is the **maximum** number of **new nodes** that could be created?*

A. 2 B. 3 C. 4 D. 5

Problem 251 (MCQ) *In a B+ tree, the maximum number of keys in a node is 5. What is the **minimum** number of keys in any **non-root** node?*

A. 1 B. 2 C. 3 D. 4

Problem 252 (MCQ) *In a B+ tree, when a **leaf node splits**, what happens to the middle key?*

A. It is moved up to the parent and removed from the leaf.
B. It is copied up to the parent and also retained in the right leaf.
C. It is copied up to the parent and retained in the left leaf.
D. It is discarded.

Problem 253 (MCQ) *In a B+ tree, when an **internal node splits**, what happens to the middle key?*

A. It is copied to the parent and retained in the left child.
B. It is copied to the parent and retained in the right child.
C. It is moved up to the parent and not retained in either child.
D. It is discarded.

Problem 254 (NAT) *A B+ tree has block size $B = 1024$ bytes, key $K = 14$ bytes, block pointer $Pb = 6$ bytes. An internal node also stores a 4-byte integer to track the number of active keys. What is the **maximum order** n (max children) of an internal node?*

Problem 255 (MCQ) *In a B+ tree, search key = 10 bytes, block pointer = 8 bytes, block size = 512 bytes. What is the **maximum** number of keys in an **internal** node?*

A. 25 B. 28 C. 29 D. 56

Problem 256 (NAT) *A B+ tree leaf node stores (key, RID) pairs where RID = 8 bytes and key = 12 bytes. Block size = 1024 bytes with 8 bytes reserved for the next-leaf pointer (unspanned). What is the **maximum** number of (key, RID) pairs a leaf node can hold?*

Problem 257 (MCQ) *Which of the following is true about the **height** of a B+ tree compared to a B-tree indexing the same r records?*

A. B+ tree is always taller because leaf nodes store record pointers.
B. B+ tree is shorter because internal nodes have higher fan-out.
C. Both trees have the same height for the same order n.
D. B-tree is shorter because data can be found at any level.

Problem 258 (NAT) *A B+ tree is used with block size $B = 1024$ bytes, $K = 12$ bytes, $Pb = 6$ bytes, $Pr = 8$ bytes, $P_{next} = 6$ bytes, indexing $r = 5,00,000$ records. Find the minimum number of disk accesses to retrieve any record.*

Problem 259 (MSQ) *Which of the following are true about **range queries** in a B+ tree? (Select all that apply.)*

A. Once the starting leaf is found, remaining records are accessed via P_{next}.
B. Range queries require traversing back to the root for each record.
C. Range queries are faster in B+ trees than in B-trees.
D. The leaf level forms a sorted linked list enabling sequential access.

Problem 260 (NAT) *In a B+ tree of order $n = 4$ (max children), keys are inserted in the following order: 10, 20, 5, 15, 30, 25, 3. How many **leaf node splits** occur during all insertions?*

7.7 GATE PYQs

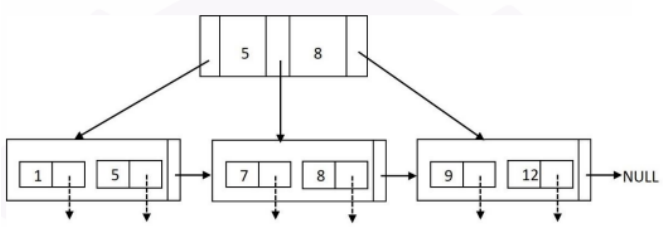
GATEPYQ 126 *In a relational database, a B+ Tree Index is to be constructed for a relation on a key field. In a B+ Tree, a Node Pointer points to a sub-tree and a Data Record Pointer points to a block of database records.*

Let, Node size = 4096 bytes, Node Pointer size = 10 bytes, Search Key Field size = 11 bytes and Data Record Pointer size = 12 bytes.

The maximum number of Node Pointers that can be present in a non-leaf node of the B+ Tree is _____

GATE DA 2026

GATEPYQ 127 *Consider a B+ Tree where the maximum number of key values in each leaf node is 2 and the maximum number of pointers in each non-leaf node is 3. Let the content of the B+ Tree be as shown in the figure.*



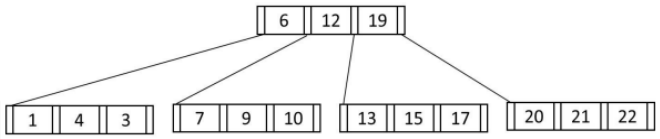
Which of the following options denotes the key value(s) stored in the root node after inserting a key value 3 in the given B+ Tree? ***GATE DA 2026***

(A) 5
(B) 8
(C) 3 and 5
(D) 3, 5 and 8

GATEPYQ 128 *An index in a DBMS is said to be dense if an index entry appears for every search-key value in the indexed file. Otherwise it is called a sparse index. Consider the following two statements.*

S1: A hash index must be a dense index
S2: A B^+ tree index can be a sparse index
Which one of the following options is correct? GATE CSE 2026
(A) Both S1 and S2 are true
(B) Both S1 and S2 are false
(C) S1 is true and S2 is false
(D) S1 is false and S2 is true

GATEPYQ 129 *Consider the following B^+ tree with 5 nodes, in which a node can store at most 3 key values.*



The value 23 is now inserted in the B^+ tree. Which of the following options(s) is/are CORRECT? GATE CSE 2025
(A) None of the nodes will split.
(B) At least one node will split and redistribute.
(C) The total number of nodes will remain same.
(D) The height of the tree will increase

GATEPYQ 130 *In a B^+- tree where each node can hold at most four key values, a root to leaf path consists of the following nodes:*

A = (49, 77, 83, -), B = (7, 19, 33, 44), C = (20, 22*, 25*, 26*)*

*The *-marked keys signify that these are data entries in a leaf.*

Assume that a pointer between keys k_1 and k_2 points to a subtree containing keys in $[k_1, k_2)$, and that when a leaf is created, the smallest key in it is copied up into its parent.

A record with key value 23 is inserted into the B^+- tree.

The smallest key value in the parent of the leaf that contains 25 is _____ GATE CSE 2025*

GATEPYQ 131 *Which of the following file organizations is/are I/O efficient for the scan operation in DBMS?*

A. Sorted

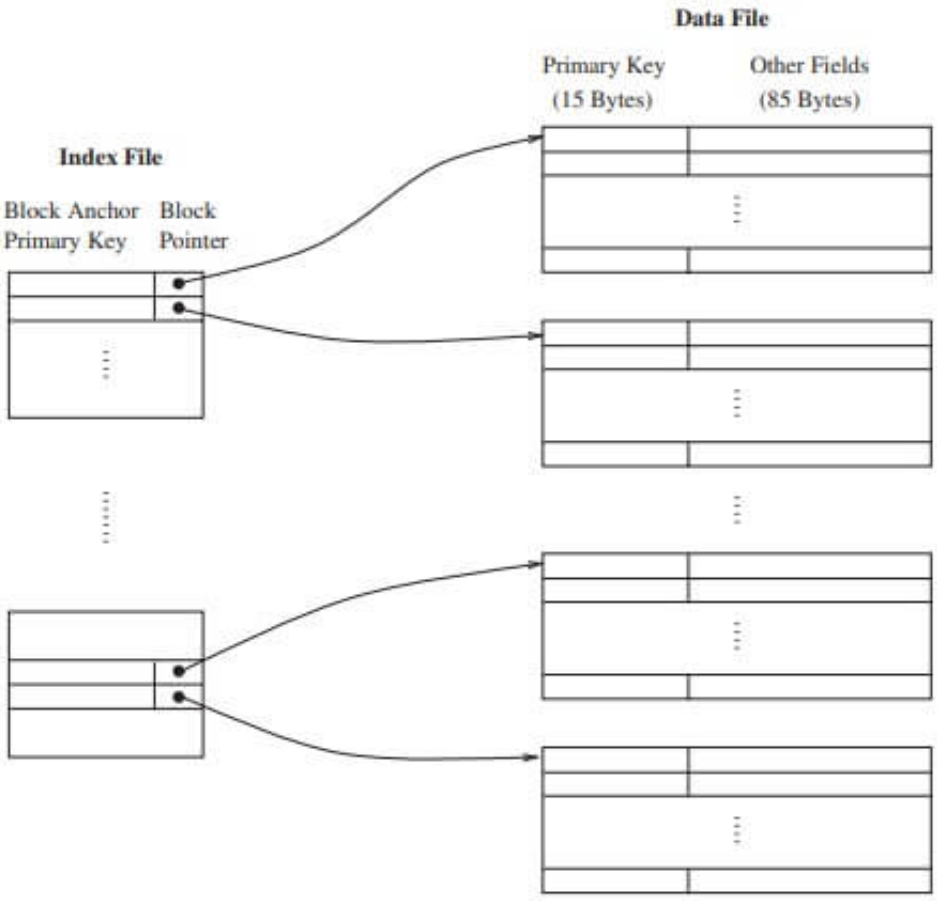
B. Heap

C. Unclustered tree index

D. Unclustered hash index

GATE CSE 2024

GATEPYQ 132 *Consider a database of fixed-length records, stored as an ordered file. The database has 25,000 records, with each record being 100 bytes, of which the primary key occupies 15 bytes. The data file is block-aligned in that each data record is fully contained within a block. The database is indexed by a primary index file, which is also stored as a block-aligned ordered file. The figure below depicts this indexing scheme.*



Suppose the block size of the file system is 1024 bytes, and a pointer to a block occupies 5 bytes. The system uses binary search on the index file to search for a record with a given key. You may assume that a binary search on an index file of b blocks takes $\lceil \log_2 b \rceil$ block accesses in the worst case.

Given a key, the number of block accesses required to identify the block in the data file that may contain a record with the key, in the worst case, is _____

GATE CSE 2023

GATEPYQ 133 *A data file consisting of 1,50,000 student-records is stored on a hard disk with block size of 4096 bytes. The data file is sorted on the primary key RollNo. The size of a record pointer for this disk is 7 bytes. Each student-record has a candidate key attribute called ANum of size 12 bytes. Suppose an index file with records consisting of two fields, ANum value and the record pointer to the corresponding student record, is built and stored on the same disk. Assume*

that the records of data file and index file are not split across disk blocks. The number of blocks in the index file is _____

GATE CSE 2021

GATEPYQ 134 *Consider a database implemented using B+ tree for file indexing and installed on a disk drive with block size of 4 KB. The size of search key is 12 bytes and the size of tree/disk pointer is 8 bytes. Assume that the database has one million records. Also assume that no node of the B+ tree and no records are present initially in main memory. Consider that each record fits into one disk block. The minimum number of disk accesses required to retrieve any record in the database is _____*

GATE CSE 2020

GATEPYQ 135 *Which one of the following statements is NOT correct about the B^+ tree data structure used for creating an index of a relational database table?*

A. B^+ Tree is a height-balanced tree

B. Non-leaf nodes have pointers to data records

C. Key values in each node are kept in sorted order

D. Each leaf node has a pointer to the next leaf node

GATE CSE 2019

GATEPYQ 136 *Consider a B^+ tree in which the search key is 12 bytes long, block size is 1024 bytes, record pointer is 10 bytes long and block pointer is 8 bytes long. The maximum number of keys that can be accommodated in each non-leaf node of the tree is _____*

GATE CSE 2015

GATEPYQ 137 *A file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index. Then that index is called*

A. Dense

B. Sparse

C. Clustered

D. Unclustered

GATE CSE 2015

GATEPYQ 138 *An index is clustered, if*

A. it is on a set of fields that form a candidate key.

B. it is on a set of fields that include the primary key

C. the data records of the file are organized in the same order as the data entries of the index.

D. the data records of the file are organized not in the same order as the data entries of the index.

GATE CSE 2013

GATEPYQ 139 *Consider a file of 16384 records. Each record is 32 bytes long and its key field is of size 6 bytes. The file is ordered on a non-key field, and the file organization is unspanned. The file is stored in a file system with block size 1024 bytes, and the size of a block pointer is 10 bytes. If the secondary index is built on the key field of the file, and a multi-level index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multi-level index are respectively*

A. 8 and 0

B. 128 and 6

C. 256 and 4

D. 512 and 5

GATE CSE 2008

GATEPYQ 140 *A clustering index is defined on the fields which are of type*

A. non-key and ordering

B. non-key and non-ordering

C. key and ordering

D. key and non-ordering

GATE CSE 2008

GATEPYQ 141 *The order of a leaf node in a B^+ tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?*

A. 63

B. 64

C. 67

D. 68

GATE CSE 2007

GATEPYQ 142 *Which one of the following is a key factor for preferring B^+-trees to binary search trees for indexing database relations?*

A. Database relations have a large number of records

B. Database relations are sorted on the primary key

C. B^+-trees require less memory than binary search trees

D. Data transfer from disks is in blocks

GATE CSE 2005

GATEPYQ 143 *The order of an internal node in a B^+ tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node?*

A. 24

B. 25

C. 26

D. 27

GATE CSE 2004

GATEPYQ 144 *A B^+-tree index is to be built on the Name attribute of the relation STUDENT. Assume that all student names are of length 8 bytes, disk blocks are of size 512 bytes, and index pointers are of size 4 bytes. Given this scenario, what would be the best choice of the degree (i.e. the number of pointers per node) of the B^+-tree?*

A. 16

B. 42

C. 43

D. 44

GATE CSE 2002

GATEPYQ 145 *There are five records in a database.*

Name	*Age*	*Occupation*	*Category*
$Rama$	27	CON	A
$Abdul$	22	ENG	A
$Jennifer$	28	DOC	B
$Maya$	32	SER	D
Dev	24	MUS	C

There is an index file associated with this and it contains the values 1, 3, 2, 5 and 4. Which one of the fields is the index built from?

A. Age
B. Name
C. Occupation
D. Category
GATE CSE 1998

7.8 Try it Yourself

Exercise 211 *Consider a database with 1,00,000 records, each of size 400 bytes, stored in an ordered file. The block size is 2048 bytes and the file is unspanned (block-aligned). A primary index is built on this file where each index entry consists of a 15-byte key and a 5-byte block pointer. If the index file itself must be block-aligned and unspanned, what is the maximum number of block accesses required to search for a key using binary search on the index file?*

Exercise 212 *A B^+ tree is used to index a relation. The block size is 4 KB. The search key is 20 bytes, the block pointer is 8 bytes, and the record pointer is 10 bytes. The tree nodes are block-aligned. Let M be the maximum number of keys in an internal node and L be the maximum number of keys in a leaf node. What is the value of $M + L$?*

Exercise 213 *A student table is stored as a heap file with 2,00,000 records. Block size is 1024 bytes and record size is 100 bytes (unspanned). A secondary index is built on a candidate key 'Aadhaar_No'. The index entry is 16 bytes (key + record pointer) and is unspanned. If binary search is used on the index file, what is the total number of block accesses required to fetch the record in the worst case?*

Exercise 214 *Consider a clustering index built on a non-key ordering attribute. The data file has 1,000 blocks and 40,000 records. There are 500 distinct values for the ordering attribute. If each index entry (attribute value + block pointer) is 20 bytes and the index file is unspanned (block size 1 KB), how many blocks does the clustering index occupy?*

Exercise 215 *Which of the following is/are **TRUE** regarding B^+ tree indexing? A. The number of leaf nodes accessed in a range query $[K_1, K_2]$ is always $\lceil (K_2 - K_1)/L \rceil$, where L is the number of keys per leaf.*
B. The internal nodes of a B^+ tree do not store record pointers.
C. Deleting a key from a B^+ tree may decrease the height of the tree only if the root becomes empty.
D. For a fixed block size, B^+ trees typically have a higher height than B trees for the same number of records.

Exercise 216 *An index-organized table (clustered index) stores records of size 512 bytes directly in the leaf nodes of a B^+ tree. Block size is 2048 bytes. Key size is 16 bytes and block pointer is 8 bytes. If the tree is exactly 3 levels deep (root is Level 1) and all nodes are 100% full, what is the maximum number of records the table can store? (Assume leaf nodes have a 8-byte next-leaf pointer).*

Exercise 217 *Suppose we have a multi-level index where the data file has 10^6 blocks. Each index block can hold 100 pointers. If we want the top level of the index to fit into exactly **one** block, how many levels of index are required, and how many blocks are in the level just above the data file?*

Exercise 218 *In a B^+ tree of order n (maximum pointers in an internal node), what is the **minimum** number of keys in a non-root internal node and a leaf node respectively? A. $\lceil n/2 \rceil$, $\lceil n/2 \rceil$*
B. $\lceil n/2 \rceil - 1$, $\lfloor n/2 \rfloor$
C. $\lfloor n/2 \rfloor$, $\lceil n/2 \rceil - 1$
D. $\lceil n/2 \rceil - 1$, $\lceil (n-1)/2 \rceil$

Exercise 219 *A file has 16,384 records of size 64 bytes each, stored unspanned in blocks of 512 bytes. A secondary index on a non-key attribute is built. There are 1024 distinct values for this attribute, and records are not sorted by this attribute. If the index is sparse (one entry per distinct value), and each entry is 12 bytes, what is the number of index blocks?*

Exercise 220 *Consider a disk with block size $B = 512$ bytes. We use a B tree (not B^+) of order p. Each node contains $(p-1)$ keys, p block pointers, and $(p-1)$ record pointers. Key size is 10 bytes, block pointer is 6 bytes, and record pointer is 4 bytes. What is the maximum possible value of p?*

Exercise 221 *Which of the following operations on a linear-list directory **cannot** be optimized using a hash index? A. Checking for filename uniqueness during file creation.*
B. Searching for a file metadata by name.
C. Deleting a file by name.
D. Listing all files in the directory in alphabetical order.

Exercise 222 *A file is stored in 5,000 blocks. We build a primary index where each index block holds 100 entries. If the index is multi-level, what is the **total** number of blocks used by the index levels?*

Exercise 223 *In a B^+ tree, if the block size is 1 KB, search key is 12 bytes, and block pointer is 8 bytes, what is the maximum number of keys that can be accommodated in a leaf node? Assume each entry in the leaf is a (key, record pointer) pair, record pointer is 10 bytes, and there is an 8-byte pointer to the next leaf node.*

Exercise 224 *A B^+ tree has height $h = 3$. The internal node fan-out is 50 and the leaf node capacity is 20 records. What is the **minimum** number of records in the database if the root has 2 children and all other nodes satisfy the minimum occupancy constraint?*

Exercise 225 *Consider a heap file. A secondary index is built on a non-key attribute. If a record is updated such that the indexed attribute value changes, which of the following is true? A. Only the data file needs to be updated.*
B. Only the index entry needs to be updated.
C. Both the data file and the index file must be updated, and the index entry may need to be moved.
D. The index remains valid as long as the record's RID does not change.

7.9 YouTube Links and QR Codes

Lecture	Details	YouTube Link	QR Code
48	Record Organization — Record Format, Spanned vs Unspanned, Blocking Factor	https://youtu.be/uYfAf9z03Yo	
49	Indexing in DBMS — Primary, Clustering, Secondary, Dense, Sparse, Multilevel	https://youtu.be/mAYmSqMluFc	
50	B-Tree Indexing — Dynamic Multilevel Indexing Explained	https://youtu.be/M5_i467KpoM	
51	B+ Tree Indexing — Structure, Insertion, Searching	https://youtu.be/YocLPtTjXmI	

52	Problem Solving on Record Organization & Indexing	https://youtu.be/pQOsn75d3u0	
53	Solved GATE PYQs on Record Organization & Indexing	https://youtu.be/BDTNPLtW4nE	

Chapter 8

Transaction Management and Concurrency Control

8.1 Transaction Management

Transaction

A transaction is a set of logically related database operations that are executed as a single unit of work. It represents a complete task performed by a user or application program to access or modify the database.
A transaction may consist of one or more read and write operations and is treated as an atomic execution block by the DBMS. The system ensures that either all operations of the transaction are successfully completed, or none of them are reflected in the database.
Key characteristics:

- A transaction groups related operations together.
- It is executed by a single user or program unit.
- It transforms the database from one consistent state to another consistent state.
- Partial execution results must not be permanently stored.

A transaction is internally executed using two fundamental database operations:

- **Read(X):** Reads the value of data item X from the database and copies it into a buffer in main memory.
- **Write(X):** Writes the updated value of data item X from the buffer back to the database.

Transaction Execution Model

During execution, a transaction operates on data items through a memory buffer:

- Data is first brought from disk to memory using **Read**.
- All computations are performed on the buffer copy.
- The final updated value is stored back using **Write**.
- Until a commit occurs, database changes are not considered permanent.

Example 314: Debit Transaction

Consider a debit transaction on a bank account balance X consisting of the following operations:

1. `R(X);`
2. `X = X - 500;`
3. `W(X);`

Assume the initial value of $X = 4000$.
Execution steps:

- After `R(X)` $\rightarrow$ Buffer contains 4000.
- After computation $\rightarrow$ Buffer becomes 3500.
- After `W(X)` $\rightarrow$ Database value becomes 3500.

If a failure occurs after step 2 but before step 3, the database still contains 4000 while computation has already deducted 500. This leads to inconsistency unless recovery control is applied.

Transaction Failure and Need for Recovery

A transaction may fail before completing all its operations due to:

- Hardware failure
- Software crash
- Power outage
- System errors

If a failure occurs mid-execution, partial updates must not be reflected in the database. Therefore, DBMS provides control operations to ensure correctness and recoverability.

Commit and Rollback Operations

Commit:

- Indicates successful completion of a transaction.
- All updates performed by the transaction are permanently stored in the database.
- After commit, changes cannot be undone by the system.

Rollback (Abort):

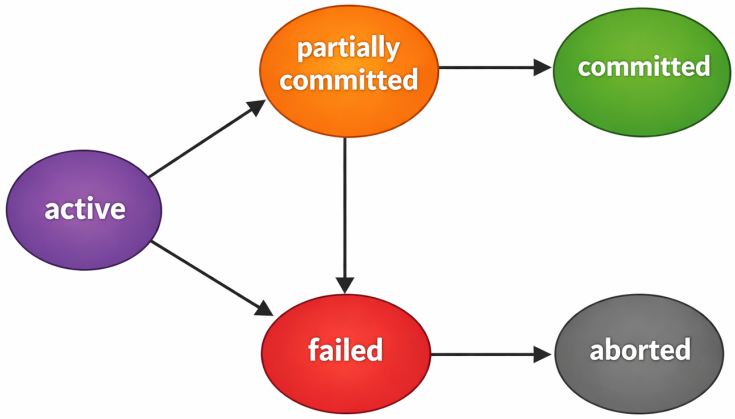
- Used when a transaction fails or is cancelled.
- All updates made by the transaction are undone.
- Database is restored to the state before the transaction began.

8.1.1 State Diagram

Transaction States

During its lifecycle, a database transaction passes through a sequence of well-defined states. These states are maintained by the transaction manager and recovery manager to ensure correctness, atomicity, and recoverability of database operations.

At any instant, a transaction is in exactly one of the defined states. State transitions occur based on execution progress, validation checks, commit actions, or failure events.



Types of Transaction States

Active:

- The transaction is currently executing its read and write operations.
- This is the initial state after the transaction begins.
- The transaction remains active until its last statement is executed or an error occurs.

Partially Committed:

- The transaction has executed its final operation.
- Updates are still in buffers/logs and not yet permanently written to the database.
- System checks (logging, constraints, commit record write) are still pending.

Committed:

- The transaction has successfully completed all operations.
- Commit record is written to stable storage.
- All effects become permanent in the database.
- No rollback is required after this state.

Failed:

- The transaction cannot proceed due to an error or system failure.
- Causes include constraint violation, deadlock victim selection, crash, or explicit abort.
- Further execution of the transaction is stopped.

Aborted:

- All effects of the failed transaction are undone using rollback.
- Database is restored to the state before the transaction began.
- After abort, the system may either restart the transaction or permanently terminate it.

8.1.2 ACID Properties

ACID Properties

ACID properties define the correctness guarantees that a DBMS must provide for every transaction. They ensure reliability, correctness, and recoverability of database operations even in the presence of failures and concurrent execution.
ACID stands for Atomicity, Consistency, Isolation, and Durability. Every well-formed transaction executed by the DBMS is expected to satisfy all four properties simultaneously.

Atomicity

Atomicity ensures that a transaction is treated as an indivisible unit of execution.

- Either all operations of the transaction are executed successfully, or none are applied.
- Partial updates are never allowed to remain in the database.
- If any operation fails, the entire transaction is rolled back.
- Implemented using undo logging and rollback mechanisms.

Formally: For transaction T, database state changes occur as **all-or-nothing**.

Consistency

Consistency ensures that a transaction preserves all integrity constraints of the database.

- A transaction transforms the database from one valid (consistent) state to another valid state.
- Constraints such as keys, domain rules, referential integrity, and business rules must hold after commit.
- Any transaction that violates constraints is aborted.

Consistency is enforced jointly by:

- Constraint definitions
- Application logic
- DBMS validation checks

Isolation

Isolation guarantees that concurrent transactions execute without interfering with each other.

- Each transaction behaves as if it is executing alone in the system.
- Intermediate results of an uncommitted transaction are not visible to other transactions.
- Prevents anomalies such as dirty reads, non-repeatable reads, and lost updates.
- Achieved using concurrency control techniques such as locking, timestamp ordering, and MVCC.

Result: Concurrent execution is equivalent to some serial execution order (serializability).

Durability

Durability guarantees that once a transaction commits, its effects are permanent.

- Committed updates survive system crashes and power failures.
- Changes are written to stable (non-volatile) storage.

- Recovery mechanisms such as write-ahead logging and checkpoints ensure persistence.

After commit, changes must not be lost under any failure scenario supported by the recovery system.

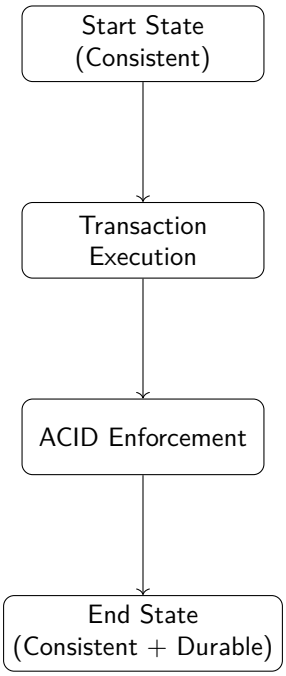
Example 315: ACID Illustration — Bank Transfer

Consider a fund transfer transaction that moves 500 from account A to account B:

1. Read(A)
2. $A = A - 500$
3. Write(A)
4. Read(B)
5. $B = B + 500$
6. Write(B)
7. Commit

ACID interpretation:

- **Atomicity:** Either both debit and credit occur, or neither occurs.
- **Consistency:** Total money $A + B$ remains unchanged.
- **Isolation:** Other transactions cannot see intermediate values after only debit.
- **Durability:** After commit, updated balances persist even after crash.



8.1.3 Serializability

Schedule

A schedule is an ordered sequence of operations from one or more transactions that preserves the program order of operations within each individual transaction.
It represents how the DBMS actually interleaves the read, write, commit, and abort operations of concurrent transactions during execution.
Key points:

- A schedule is also called a history of execution.
- The relative order of operations inside each transaction must be preserved.
- Operations of different transactions may be interleaved.
- A schedule may be serial or non-serial.

Types of Schedules

Serial Schedule:

- One transaction completes fully before the next transaction starts.
- No interleaving of operations occurs.
- Always correct and consistent.
- Poor concurrency and low throughput.

Non-Serial (Concurrent) Schedule:

- Operations of multiple transactions are interleaved.
- Provides better concurrency and resource utilization.
- May or may not preserve database consistency.
- Must be tested for serializability.

Example 316: Serial and Non-Serial Schedule

Serial Schedule

T1	T2
READ1(A)	
WRITE1(A)	
READ1(B)	
C1	
	READ2(B)
	WRITE2(B)
	READ2(B)
	C2

Non-Serial Schedule

T1	T2
READ1(A)	
WRITE1(A)	
	READ2(B)
	WRITE2(B)
READ1(B)	
WRITE1(B)	
READ1(B)	

Concurrency Issues in Non-Serial Schedules

When transactions are interleaved without proper control, several anomalies can occur that violate correctness.

Major Concurrency Anomalies

Lost Update:

- Two transactions update the same data item.
- One update overwrites the other.
- Final value reflects only one transaction.

Dirty Read (Uncommitted Dependency):

- A transaction reads data written by another uncommitted transaction.
- If the writer aborts, the reader has used invalid data.

Non-Repeatable Read:

- A transaction reads the same item twice and gets different values.
- Another committed transaction updated the value in between.

Incorrect Summary / Inconsistent Retrieval:

- Aggregate computation reads some old and some new values.
- Occurs when updates happen during summary calculation.

Serializability

Serializability is the correctness criterion for concurrent schedules. A non-serial schedule is serializable if its net effect on the database is equivalent to some serial schedule.
It ensures that concurrent execution preserves database consistency exactly as a serial execution would.

Conflicting Operations

Two operations conflict if all the following conditions hold:

1. They belong to different transactions.
2. They access the same data item (shared).
3. At least one of them is a WRITE operation.

Conflicting pairs on data item a:

- READ(a) – WRITE(a)
- WRITE(a) – READ(a)
- WRITE(a) – WRITE(a)

Types of Serializability

Conflict Serializability:

- A schedule is conflict serializable if it can be converted to a serial schedule by swapping only non-conflicting operations.
- Tested using a precedence (serialization) graph.
- Easy to check algorithmically.

View Serializability:

- Based on equivalence of read-from relations and final writes.
- A schedule is valid if it produces the same final view as some serial schedule.
- More general than conflict serializability.
- Hard to test in practice.

8.1.3.1 Conflict Serializability

Conflict Serializability

A non-serial schedule is said to be **conflict serializable** if it can be transformed into a serial schedule by repeatedly swapping adjacent non-conflicting operations without changing the effect of the schedule.
Two schedules are conflict equivalent if:

- They contain the same operations of the same transactions, and
- The relative order of every pair of conflicting operations is the same in both schedules.

Conflict serializability is the most widely used practical test for correctness of concurrent schedules because it can be verified using a precedence (serialization) graph.

Detailed Procedure to Test Conflict Serializability

Given a schedule S:

1. List all operations in exact execution order.
2. Identify all **conflicting pairs**:
   - Different transactions
   - Same data item
   - At least one WRITE
3. Draw a **precedence graph**:
   - One node per transaction.
   - For each conflicting pair where T_i's operation appears before T_j's operation, add edge $T_i \to T_j$.
4. Check the graph:
   - If the graph has a cycle $\to$ Not conflict serializable.
   - If acyclic $\to$ Conflict serializable.
5. Apply topological sort to obtain equivalent serial order(s).

Example 317: Two Transactions — Serializable

Schedule:

$$R_1(A)\ W_1(A)\ R_2(A)\ W_2(A)$$

Step 1: Conflicts on A:

- $W_1(A)$ before $R_2(A) \to$ conflict
- $W_1(A)$ before $W_2(A) \to$ conflict

Step 2: Edges:

$$T_1 \to T_2$$

Step 3: Graph:

No cycle $\to$ Conflict serializable.
Equivalent serial order: T_1, T_2.

Example 318: Two Transactions — Not Serializable

Schedule:

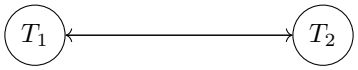
$$R_1(A)\ W_2(A)\ W_1(A)\ R_2(A)$$

Conflicts:

- $R_1(A)$ before $W_2(A) \to T_1 \to T_2$

- $W_2(A)$ before $W_1(A) \rightarrow T_2 \rightarrow T_1$

Graph:



Cycle exists $\rightarrow$ Not conflict serializable.

Example 319: Three Transactions — Serializable

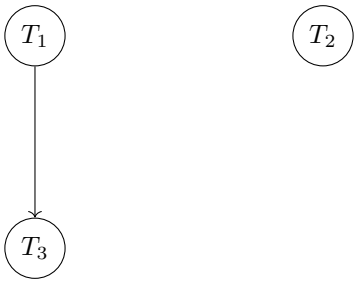
Schedule:

$$R_1(A)\ W_1(A)\ R_2(B)\ W_2(B)\ R_3(A)\ W_3(A)$$

Conflicts:

- $W_1(A)$ before $R_3(A) \rightarrow T_1 \rightarrow T_3$
- $W_1(A)$ before $W_3(A) \rightarrow T_1 \rightarrow T_3$

Graph:



No cycle $\rightarrow$ Serializable.
Possible serial orders: T_1, T_2, T_3 or T_2, T_1, T_3.

Example 320: Three Transactions — Not Serializable

Schedule:

$$W_1(A)\ R_2(A)\ W_2(B)\ R_3(B)\ W_3(A)$$

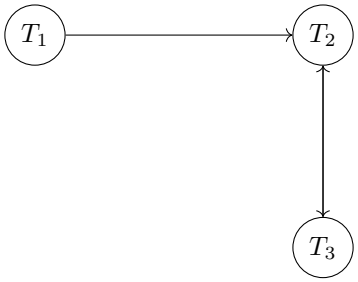
Conflicts:

- $W_1(A)$ before $R_2(A) \rightarrow T_1 \rightarrow T_2$
- $W_2(B)$ before $R_3(B) \rightarrow T_2 \rightarrow T_3$
- $R_2(A)$ before $W_3(A) \rightarrow T_2 \rightarrow T_3$
- $W_3(A)$ after $W_1(A)$ gives $T_1 \rightarrow T_3$

Add also conflict:

$$W_3(A) \text{ vs } R_2(A) \Rightarrow T_3 \rightarrow T_2$$

Graph contains cycle between T_2 and T_3.



Cycle $\rightarrow$ Not conflict serializable.

Example 321: Three Transactions — Serializable with Multiple Orders

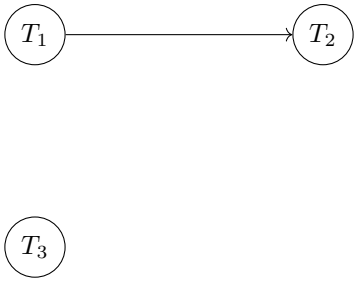
Schedule:

$$R_1(A)\ W_1(A)\ R_2(A)\ R_3(B)\ W_3(B)$$

Conflicts:

$$W_1(A) \rightarrow R_2(A) \Rightarrow T_1 \rightarrow T_2$$

No other shared-item writes.
Graph:



Acyclic $\rightarrow$ Serializable.
Valid serial orders include:

$$(T_1, T_3, T_2),\ (T_3, T_1, T_2)$$

Example 322: Four Transactions — Serializable

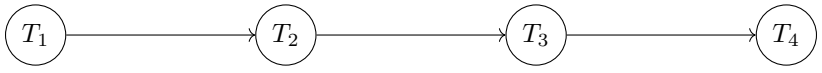
Schedule:

$$W_1(A)\ R_2(A)\ W_2(B)\ R_3(B)\ W_3(C)\ R_4(C)$$

Conflicts:

- $W_1(A)$ before $R_2(A) \rightarrow T_1 \rightarrow T_2$
- $W_2(B)$ before $R_3(B) \rightarrow T_2 \rightarrow T_3$
- $W_3(C)$ before $R_4(C) \rightarrow T_3 \rightarrow T_4$

Graph:



No cycle $\rightarrow$ Conflict serializable.
Serial order: T_1, T_2, T_3, T_4.

Topological Sorting for Conflict Serializability

Topological sorting is used to convert a conflict-serializable non-serial schedule into its equivalent serial schedule using the precedence (serialization) graph.
If the precedence graph is acyclic, then at least one topological ordering exists. Each valid topological ordering gives one equivalent serial execution order of transactions.
Thus, topological sorting is the final step after constructing the precedence graph in conflict serializability testing.

Topological Sorting Procedure on Precedence Graph

Given a precedence graph:

1. Construct the precedence graph using conflicting operations.
2. Verify that the graph has no cycles.
3. Find a node with **in-degree = 0** (no incoming edges).
4. Output that transaction in the serial order.
5. Remove that node and all its outgoing edges from the graph.
6. Repeat the process on the reduced graph.
7. Continue until all nodes are removed.

Notes:

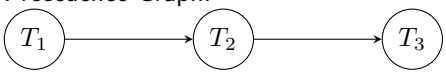
- If at any step no node has in-degree 0 and nodes remain → cycle exists → not conflict serializable.
- If multiple nodes have in-degree 0 → multiple serial orders are possible.

Example 323: 3T Acyclic Case–1

Schedule:

$$W_1(A)\ R_2(A)\ W_2(B)\ R_3(B)$$

Conflicts:
$W_1(A), R_2(A) \rightarrow T_1 \rightarrow T_2$
$W_2(B), R_3(B) \rightarrow T_2 \rightarrow T_3$
Precedence Graph:



No cycle → Serializable
Topological Order:

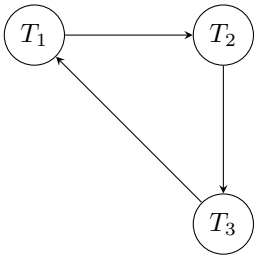
$$T_1 \rightarrow T_2 \rightarrow T_3$$

Example 324: 3T Cyclic Case

Schedule:

$$W_1(A)\ R_2(A)\ W_2(B)\ R_3(B)\ W_3(A)$$

Conflicts:
$W_1(A), R_2(A) \rightarrow T_1 \rightarrow T_2$
$W_2(B), R_3(B) \rightarrow T_2 \rightarrow T_3$
$W_3(A), W_1(A) \rightarrow T_3 \rightarrow T_1$
Graph:



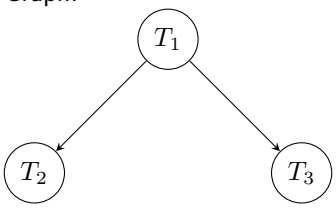
Cycle exists → NOT conflict serializable
Topological sorting not possible.

Example 325: 3T Multiple Serial Orders

Schedule:

$$W_1(A)\ R_2(A)\ W_1(B)\ R_3(B)$$

Conflicts:
$W_1(A), R_2(A) \rightarrow T_1 \rightarrow T_2$
$W_1(B), R_3(B) \rightarrow T_1 \rightarrow T_3$
Graph:



Acyclic → Serializable
Topological Orders:

$$T_1, T_2, T_3$$

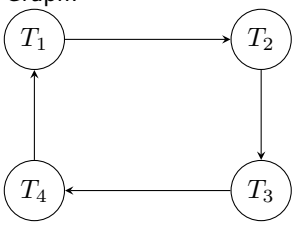
$$T_1, T_3, T_2$$

Example 326: 4T Cyclic Case

Schedule:

$$W_1(A)\ R_2(A)\ W_2(B)\ R_3(B)\ W_3(C)\ R_4(C)\ W_4(A)$$

Conflicts:
$T_1 \rightarrow T_2$
$T_2 \rightarrow T_3$
$T_3 \rightarrow T_4$
$T_4 \rightarrow T_1$
Graph:



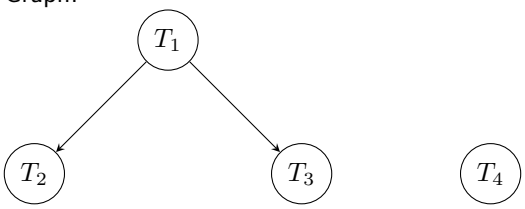
Cycle exists → Not serializable.

Example 327: 4T Acyclic Partial Order

Schedule:

$$W_1(A)\ R_2(A)\ W_1(B)\ R_3(B)\ W_4(C)$$

Conflicts:
$T_1 \to T_2$
$T_1 \to T_3$
No conflicts with T_4.
Graph:



Acyclic → Serializable
Possible Topological Orders:
T_1, T_2, T_3, T_4
T_1, T_3, T_2, T_4
T_4 can appear anywhere since independent.

8.1.3.2 View Serializability

Motivation — Why View Serializability?

Conflict serializability uses a simple rule: *if two operations conflict, their relative order must be preserved.* This rule is **sufficient** for correctness, but it is **too strict**. Two schedules may produce *identical results* even when their conflicting operations appear in a different order — as long as every read sees the same value and the database ends in the same state.

View serializability captures exactly this broader idea. It asks:

"Do both schedules tell the same data-flow story?"

A schedule is **view serializable** if it is view equivalent to *some* serial schedule.

Relationship to conflict serializability:

$$\text{Conflict Serializable} \subset \text{View Serializable} \subset \text{All Schedules}$$

Every conflict-serializable schedule is view serializable, but the converse is *not* always true.

Intuition — Why Exactly Three Conditions?

Think of a database transaction as a **story** about values: someone reads an initial value, someone passes a computed value to the next reader, and someone writes the final answer.
Two schedules are "the same story" when *three things match*:

Condition 1 *Who reads the original value?*

Condition 2 *Who reads from whom?*

Condition 3 *Who writes the final answer?*

Each condition closes a different loophole:

- **Without Condition 1:** a schedule could silently swap which transaction depends on the pre-existing database

value, changing whose logic runs on "clean" data.

- **Without Condition 2:** a read could receive a value from a *different* writer, altering what computation is performed — even if the final write looks correct.
- **Without Condition 3:** the database could be left in a different state after all transactions finish, violating durability.

All three together guarantee **value-flow equivalence**: both schedules do the same logical work, just possibly in a different interleaving order.

Formal Definition — View Equivalence

Two schedules S_1 and S_2 on the same set of transactions are **view equivalent** ($S_1 \approx_v S_2$) if and only if *all three* of the following conditions hold simultaneously:

Condition 1 — Initial Read Preservation

> If T_i reads the initial (pre-schedule) value of data item X in S_1, then T_i must also read the initial value of X in S_2.

Ensures the same transactions depend on the original database.

Condition 2 — Read-From Preservation

> If T_i reads a value of X that was written by T_j in S_1, then T_i must read X from T_j in S_2 as well.

Preserves the producer $\rightarrow$ consumer chain.

Condition 3 — Final Writer Preservation

> For every data item X, the transaction that performs the *last write* on X in S_1 must also perform the last write on X in S_2.

Guarantees the final database state is identical.

Blind Write — Definition and Key Theorem

Definition. A transaction T_i performs a **blind write** on data item X if it writes X *without first reading* X.

$$T_i \text{ blindly writes } X \iff W_i(X) \text{ occurs in the schedule but } R_i(X) \text{ does not.}$$

Why blind writes matter for view serializability:
When T_i reads X before writing it, its written value *depends* on what it read. The three view conditions then lock down what it must have read — creating strong ordering constraints.
When T_i blindly writes X, it produces a value independently of any read. The view conditions for reads are vacuously satisfied (no read to constrain), leaving more freedom to rearrange transactions — freedom that can sometimes break a cycle.

Blind Write Theorem (Practical Shortcut).

> *If a schedule S is **not** conflict serializable and contains **no** blind writes, then S is **not** view serializable.*
>
> *Equivalently: a non-conflict-serializable schedule can be view serializable* only if *it contains at least one blind write.*

This theorem is the key shortcut in practical testing. It avoids the expensive search over all serial schedules unless blind writes are actually present.

Step-by-Step Testing Algorithm for View Serializability

Step 1 — Check Conflict Serializability
Build the conflict (precedence) graph. If acyclic ⇒ **conflict serializable** ⇒ automatically **view serializable**. **Stop.**

Step 2 — Check for Blind Writes (Only if Step 1 Fails)
If the graph has a cycle, scan each transaction for **Blind Writes** ($W(X)$ without a preceding $R(X)$).

- *No blind writes found:* The schedule is **not view serializable**. **Stop.**
- *At least one blind write found:* The schedule *might* be view serializable. Continue to Step 3.

Step 3 — Extract the Three View Constraints
For every data item X in the schedule, identify:

- Which transaction reads the *initial* value of X?
- For each read $R_i(X)$, which write immediately precedes it?
- Which transaction performs the *last* write on X?

Step 4 — Derive Ordering Constraints
Translate the view constraints into ordering requirements:

- **Rule 1:** T_i reads initial $X \Rightarrow T_i$ must be before every writer of X.
- **Rule 2:** T_i reads from $T_j \Rightarrow T_j$ must be the last writer of X before T_i.
- **Rule 3:** T_k is final writer $\Rightarrow T_k$ must be after all other writers of X.

Step 5 — Check for a Valid Serial Order

- If a linear ordering satisfies all constraints ⇒ **view serializable**.
- If constraints create a cycle ⇒ **not view serializable**.

Summary — View Serializability at a Glance

Schedule Type	Conflict Serial.	View Serial.
Serial	✓	✓
Conflict serializable	✓	✓
Blind writes (cyclic graph)	×	may or may not be
Cycle + no blind writes	×	×

Practical note: Checking view serializability in general is NP-hard (requires testing all $n!$ serial schedules), so it is *not* used directly in real concurrency control systems. The blind write theorem reduces unnecessary work but does not eliminate exponential worst-case complexity.

Example 328: All Blind Writes — View Serializable but Not Conflict Serializable

Schedule:

$$S: \ W_1(X), \ W_2(X), \ W_1(X)$$

No transaction reads X at all; every write is blind.

Step 1 — Conflict Serializability Check
Identify write–write conflicts on X:

$$W_1(X) \text{ before } W_2(X) \Rightarrow T_1 \to T_2$$

$$W_2(X) \text{ before } W_1(X) \Rightarrow T_2 \to T_1$$

The conflict graph has a cycle $T_1 \to T_2 \to T_1$. **Not conflict serializable.** Continue.

Step 2 — Blind Write Check

T_1: writes X but never reads X $\Rightarrow$ **blind write**

T_2: writes X but never reads X $\Rightarrow$ **blind write**

Blind writes exist. Theorem cannot rule out VS. Continue.

Step 3 & 4 — Extract and Translate View Constraints

- *Condition 1:* No transaction reads X at all $\Rightarrow$ vacuously satisfied (nothing to constrain).
- *Condition 2:* No read operations exist $\Rightarrow$ vacuously satisfied.
- *Condition 3:* The last write on X in S is $W_1(X)$ $\Rightarrow$ T_1 must be the final writer of X in any equivalent serial schedule $\Rightarrow$ T_2 must appear *before* T_1.

Only one ordering constraint: $T_2 \to T_1$.

Step 5 — Find Valid Serial Order
The constraint gives a single candidate:

$$T_2 \to T_1$$

Verify against all three conditions for this serial order:

Condition	Requirement	Satisfied?
1 (initial reads)	no reads at all	✓(vacuous)
2 (read-from)	no reads at all	✓(vacuous)
3 (final writer)	T_1 writes X last	✓($T_2 \to T_1$)

Conclusion: S is view equivalent to the serial schedule $T_2 \to T_1$. It is **view serializable but not conflict serializable**.

Example 329: Blind Write Present but Conditions Contradict — Not View Serializable

Schedule:

$$S: R_1(X),\ W_2(X),\ W_1(X)$$

T_1: $R_1(X)$, then $W_1(X)$ (reads before writing — *not* blind)

T_2: $W_2(X)$ only (writes without reading — **blind write**)

Step 1 — Conflict Serializability Check

$$R_1(X),\ W_2(X) \Rightarrow T_1 \to T_2$$

$$W_2(X),\ W_1(X)\ \Rightarrow\ T_2 \to T_1$$

Cycle $T_1 \to T_2 \to T_1$. **Not conflict serializable.**

Step 2 — Blind Write Check
T_2 writes X without reading it $\Rightarrow$ **blind write exists**. Cannot use the theorem to immediately reject. Continue.

Step 3 & 4 — Extract and Translate View Constraints

- *Condition 1:* $R_1(X)$ occurs before any write on X in S, so T_1 reads the *initial* value of X. In any equivalent serial schedule, T_1 must still read the initial value. This means T_1 must execute before both writers of X, i.e., before T_2.

$$\boxed{T_1 \text{ before } T_2}$$

- *Condition 2:* $R_1(X)$ reads the initial value (handled above). No other reads exist.
- *Condition 3:* The last write on X in S is $W_1(X)$, so T_1 must be the final writer. Since T_2 also writes X, T_2 must appear before T_1.

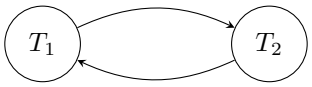
$$\boxed{T_2 \text{ before } T_1}$$

Step 5 — Check for a Valid Serial Order
The constraints require simultaneously:

$$T_1 \text{ before } T_2 \quad \textbf{and} \quad T_2 \text{ before } T_1$$

This is a direct contradiction — no linear ordering satisfies both.

Cond. 1



Cond. 3

The view dependency graph has a cycle.
Conclusion: Despite the blind write, the conditions force a contradictory ordering. S is **not view serializable**.

Example 330: Full Dependency Construction — View Serializable (also Conflict Serializable)

Schedule:

$$S:\ R_1(X),\ W_2(X),\ R_3(X),\ W_3(X)$$

T_1: $R_1(X)$ only

T_2: $W_2(X)$ only (blind write)

T_3: $R_3(X)$, then $W_3(X)$

Step 1 — Conflict Serializability Check

$R_1(X)$ vs $W_2(X)$ read–write conflict $T_1 \to T_2$

$R_1(X)$ vs $W_3(X)$ read–write conflict $T_1 \to T_3$

$W_2(X)$ vs $R_3(X)$ write–read conflict $T_2 \to T_3$

$W_2(X)$ vs $W_3(X)$ write–write conflict $T_2 \to T_3$

Conflict graph:

$$T_1 \to T_2 \to T_3 \quad \text{and} \quad T_1 \to T_3$$

Acyclic $\Rightarrow$ **conflict serializable**. Serial order: $T_1 \to T_2 \to T_3$. By the subset relationship, **view serializable** immediately.

Verifying all three view conditions explicitly for serial $T_1 \to T_2 \to T_3$:

Condition 1 (initial reads): $R_1(X)$ appears before any write in S. So T_1 reads the initial value of X. In the serial $T_1 \to T_2 \to T_3$, T_1 executes first and no one has written X yet when T_1 reads. T_1 still reads the initial value. ✓

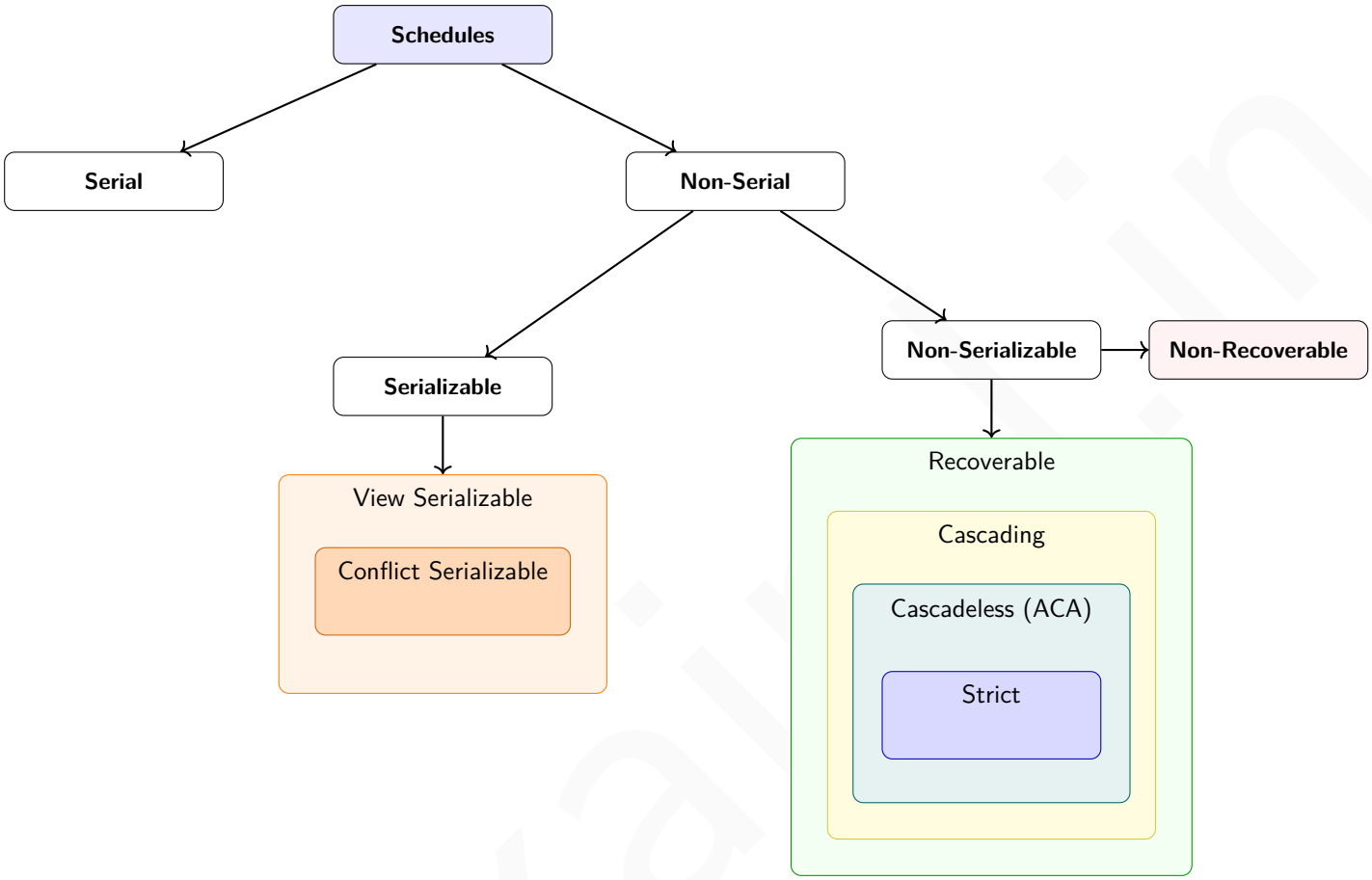
Condition 2 (read-from): $R_3(X)$ appears after $W_2(X)$ in S. The most recent write before $R_3(X)$ is $W_2(X)$, so T_3 reads from T_2. In the serial $T_1 \to T_2 \to T_3$: T_2 writes X, then T_3 reads X — it reads the value written by T_2. T_3 reads from T_2 in the serial as well. ✓

Condition 3 (final writer): The last write in S is $W_3(X)$, so T_3 is the final writer. In the serial $T_1 \to T_2 \to T_3$: T_3 executes last and writes X last. T_3 is still the final writer. ✓

Condition	Requirement	Satisfied?
1 — initial read	T_1 reads initial X	✓
2 — read-from	T_3 reads X written by T_2	✓
3 — final writer	T_3 writes X last	✓

Conclusion: All three view conditions are satisfied by the serial order $T_1 \to T_2 \to T_3$. The schedule is **view serializable** (and also conflict serializable).

8.1.4 Non Serializable Schedules



What is a Non-Serializable Schedule?

A **schedule** is an ordered sequence of operations from multiple concurrent transactions. A schedule is **serial** if all operations of one transaction execute before any operation of the next. A schedule is **serializable** if its effect is equivalent to some serial schedule.
A **non-serializable schedule** is one that *cannot* be re-ordered into any equivalent serial schedule while preserving the same final database state. Within non-serial schedules, further classification by *recovery behaviour* yields four nested categories:

- **Strict** ⊂ **Cascadeless (ACA)** ⊂ **Cascading (Recoverable)** — all inside *Recoverable*
- **Non-Recoverable** — outside all of the above

8.1.4.1 Non-Recoverable Schedules

Definition — Non-Recoverable Schedule

A schedule S is **non-recoverable** if there exist two transactions T_i and T_j such that:

1. T_i reads a data item x that was *last written by* T_j,
2. T_i *commits* before T_j commits, and
3. T_j subsequently *aborts*.

Because a committed transaction cannot be undone, the database is left in a **permanently inconsistent state** and recovery is *impossible*.

Violation rule (informal):

$$T_j \xrightarrow{\text{writes } x} T_i \xrightarrow{\text{reads } x} \underbrace{T_i \text{ COMMIT}}_{\text{permanent!}} \longrightarrow T_j \text{ ABORT} \Rightarrow \textbf{Disaster}$$

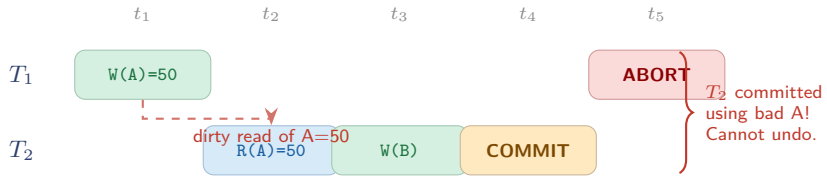
Why Non-Recoverable is the Worst Case

In every other category (cascading, ACA, strict), the database can be restored to a consistent state by rolling back transactions. In a non-recoverable schedule the committed transaction T_i has already made its changes **permanent**. No undo-log mechanism can reverse a committed transaction.

Example 331: Non-Recoverable Schedule — Worked Example

Scenario: T_1 transfers money; T_2 reads the new balance and immediately pays a bill, then commits. T_1 then discovers an error and aborts.

Timeline (time flows left → right):



Step-by-step explanation:

Time	T_1	T_2	Event
t_1	W(A=50)		T_1 writes $A = 50$ (uncommitted)
t_2		R(A=50)	T_2 reads the **dirty** value $A = 50$
t_3		W(B)	T_2 uses dirty A to compute and write B
t_4		COMMIT	T_2 **commits** — changes are permanent
t_5	ABORT		T_1 **aborts** — A must revert. But T_2 already committed!

Conclusion: The database now holds a committed value of B that was derived from an A that no longer exists. Recovery is impossible.

8.1.4.2 Cascading (Recoverable) Schedules

Definition — Recoverable Schedule

A schedule S is **recoverable** if, for every pair of transactions T_i, T_j where T_i reads a data item written by T_j:

$$\text{commit}(T_j) \text{ occurs before } \text{commit}(T_i)$$

This prevents the non-recoverable problem. However, it does *not* prevent T_j from aborting *before* it commits — which forces T_i (and any transaction that read from T_i) to abort too. This chain reaction is called a **cascading abort**.

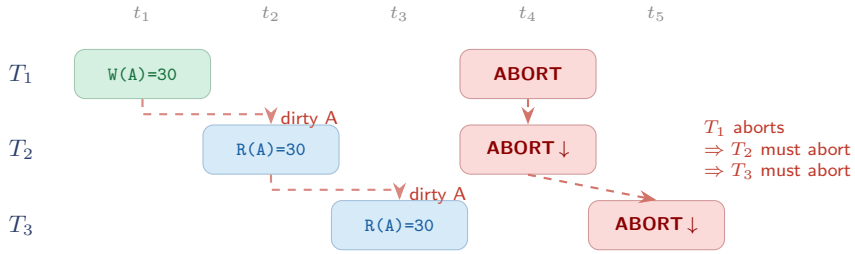
Definition — Cascading Abort

If T_j aborts, and T_i has read data written by T_j (even if that data was a dirty write), then T_i must also abort. If T_k read from T_i, then T_k must abort as well — and so on. This propagating rollback is a **cascade of aborts** and is extremely expensive in high-throughput systems.

Example 332: Cascading Abort — Worked Example

Scenario: Three transactions read a value that propagates through a chain. A single abort at the source cascades to all three.

Timeline:



Time	T_1	T_2	T_3	Event
t_1	W(A=30)			T_1 writes $A = 30$ (uncommitted)
t_2		R(A=30)		T_2 reads dirty $A = 30$
t_3			R(A=30)	T_3 reads dirty $A = 30$
t_4	ABORT	ABORT	ABORT	T_1 aborts; T_2 and T_3 *must* cascade-abort

Note: This schedule *is* recoverable (no transaction committed before its source), but cascading aborts waste enormous work.

8.1.4.3 Cascadeless / ACA Schedules

Definition — Cascadeless Schedule (ACA)

A schedule S **avoids cascading aborts (ACA)** (also called a **cascadeless schedule**) if, for every read of data item x by transaction T_i:

$$T_i \text{ reads } x \implies \text{the transaction that last wrote } x \textbf{ has already committed.}$$

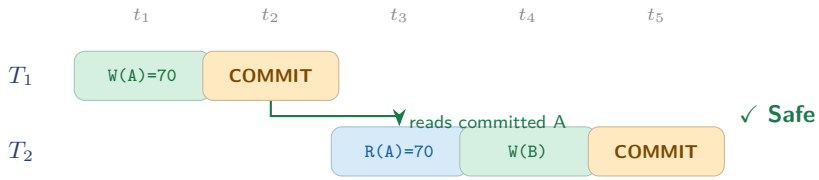
By forbidding *dirty reads*, no chain of aborts can propagate. If any transaction aborts, only that single transaction need be rolled back.

Containment: Every cascadeless schedule is also recoverable. However, cascadeless schedules may still allow **dirty writes** (overwriting data written by an uncommitted transaction), which complicates undo. Strict schedules tighten this further.

Example 333: Cascadeless Schedule — Safe Read After Commit

Scenario: T_2 waits for T_1 to commit before reading A.

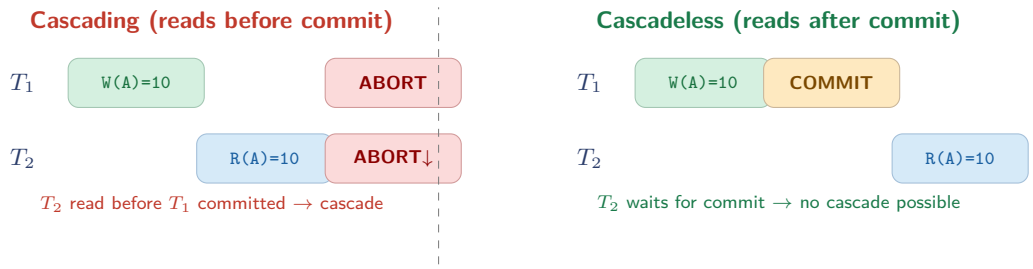
Timeline:



Time	T_1	T_2	Event
t_1	W(A=70)		T_1 writes $A = 70$ (not yet committed)
t_2	COMMIT		T_1 **commits** — $A = 70$ is now permanent
t_3		R(A=70)	T_2 reads the **committed** value — safe
t_4		W(B)	T_2 computes and writes B
t_5		COMMIT	T_2 commits cleanly

Conclusion: Even if T_1 had aborted at t_1, T_2 would never have read the dirty value. No cascade is possible.

Example 334: ACA vs Cascading Direct Comparison



8.1.4.4 Strict Schedules

Definition — Strict Schedule

A schedule S is **strict** if, for every data item x written by transaction T_j that has *not yet committed or aborted*:

- No other transaction T_i may **read** x, and
- No other transaction T_i may **write** x.

Strict schedules forbid both *dirty reads* **and** *dirty writes*. This is strictly stronger than ACA.

Why strictness simplifies recovery: To undo T_j's write of x, the DBMS need only restore the **before-image** of x (its value before T_j started). Since no other transaction has touched x since T_j's write, there is no ambiguity — the before-image is always correct.

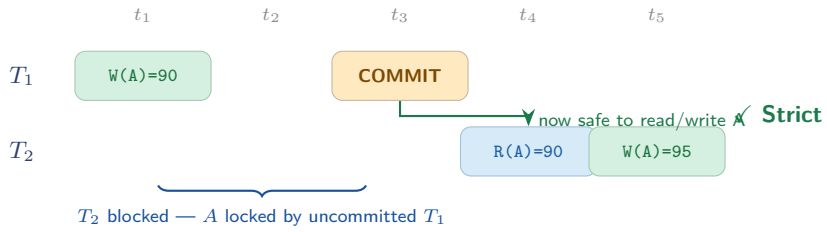
Containment:

$$\text{Strict} \subsetneq \text{Cascadeless (ACA)} \subsetneq \text{Cascading} \subsetneq \text{Recoverable}$$

Example 335: Strict Schedule — Worked Example

Scenario: T_2 needs to both read and overwrite A, which T_1 has already written. In a strict schedule, T_2 must wait until T_1 commits (or aborts) before doing either.

Timeline:



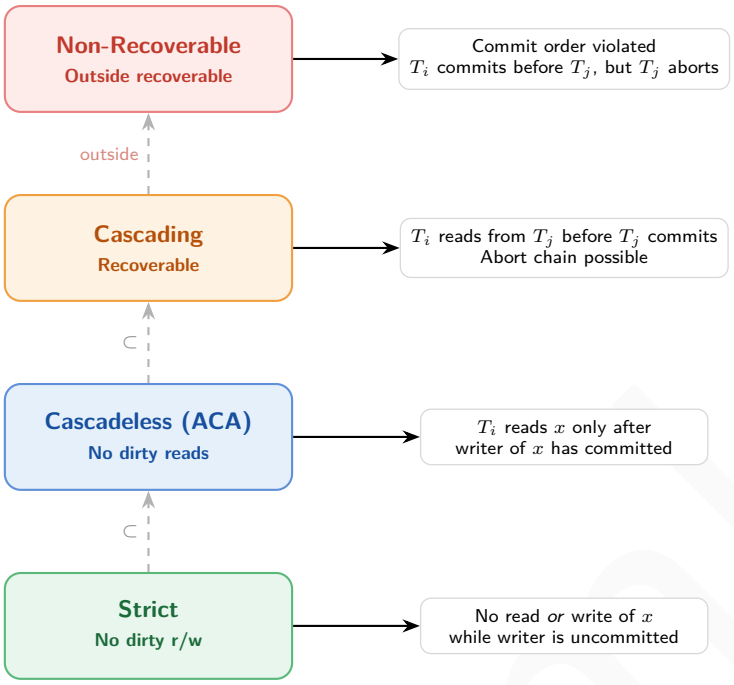
Time	T_1	T_2	Event
t_1	W(A=90)		T_1 writes $A = 90$ (uncommitted)
t_2		*(blocked)*	T_2 wants to read A — **blocked**
t_3	COMMIT		T_1 commits; $A = 90$ is permanent
t_4		R(A=90)	T_2 reads committed A — safe
t_5		W(A=95)	T_2 overwrites A — also safe

Undo simplicity: If T_1 had aborted instead of committing, restoring A to its before-image is trivial — no other transaction has touched A in the interim.

8.1.5 Comparison and Summary

Key Differentiating Rules at a Glance

Category	Dirty reads	Dirty writes	Cascade aborts	Recovery
Non-recoverable	✓ Possible	✓ Possible	✓ Possible	**Impossible**
Cascading	✓ Possible	✓ Possible	✓ Possible	Possible
Cascadeless (ACA)	× Forbidden	✓ Possible	× Impossible	Easy
Strict	× Forbidden	× Forbidden	× Impossible	Trivial



8.2 Concurrency Control

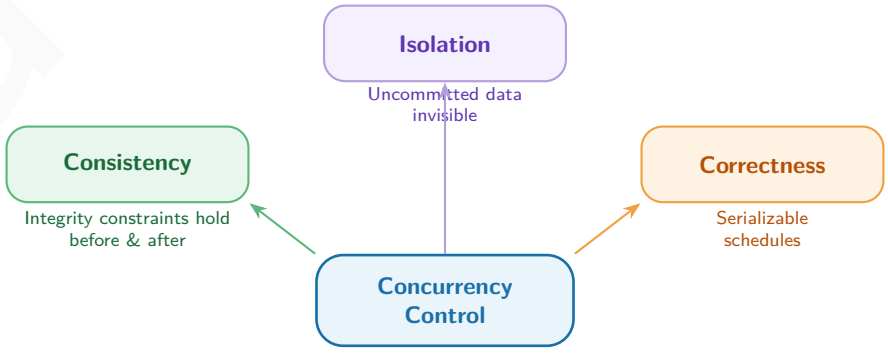
Concurrency Control in DBMS — Core Concept

Concurrency control in a DBMS coordinates simultaneous access to the database by multiple transactions. In a multi-user environment it ensures:

- The interleaved execution of transactions produces results equivalent to *some serial execution order*.
- Undesirable effects caused by uncontrolled concurrent access are **prevented**.
- Correctness and consistency of the database are guaranteed.

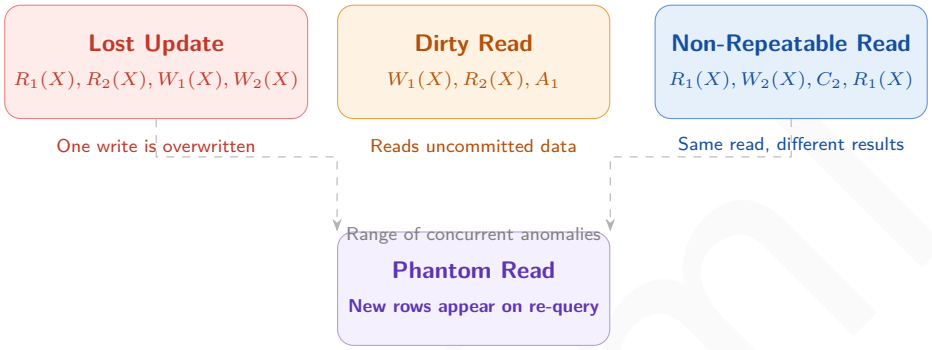
Key Objectives of Concurrency Control

- **Consistency:** The database remains in a consistent state despite concurrent execution. All integrity constraints hold before and after every transaction.
- **Isolation:** Each transaction executes as if no other transactions are running. Intermediate (uncommitted) results are invisible to others.
- **Correctness:** Conflicts are prevented and data integrity is preserved. The concurrent schedule must be *serializable*.



Challenges in Concurrent Execution

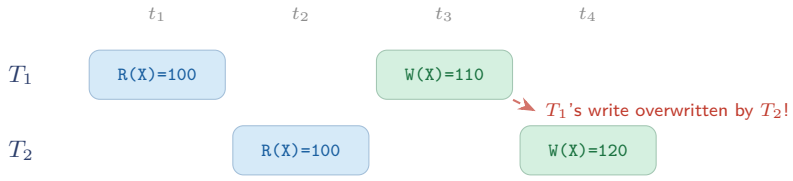
- **Lost Update:** Two transactions update the same item; one overwrites the other's change.
- **Inconsistent Retrieval:** A read sees partially updated data while another transaction is still modifying it.
- **Dirty Read (Uncommitted Data):** A transaction reads a value written by another transaction that has not yet committed.
- **Non-Repeatable Read:** The same read inside a transaction returns different values because another transaction committed an update in between.
- **Phantom Read:** A re-executed range query returns new rows inserted by another committed transaction.



Example 336: Lost Update — Worked Example

Initial value: $X = 100$

Schedule: $R_1(X);\ R_2(X);\ W_1(X{=}110);\ W_2(X{=}120)$

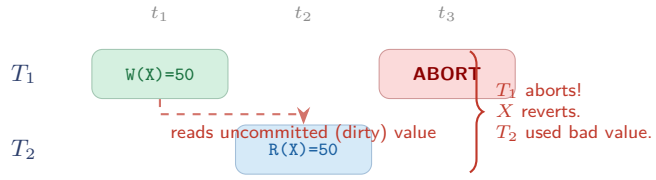


Time	T_1	T_2	Database
t_1	R(X)		$X = 100$
t_2		R(X)	$X = 100$
t_3	W(X=110)		$X = 110$
t_4		W(X=120)	$X = 120$

Problem: Correct serial result would be $X = 130$. T_1's update $(+10)$ is completely lost.

Example 337: Dirty Read — Worked Example

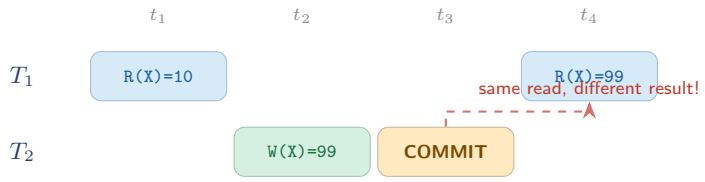
Schedule: $W_1(X);\ R_2(X);\ A_1$



Problem: T_2 based its work on $X = 50$, but that value never permanently existed.

Example 338: Non-Repeatable Read — Worked Example

Schedule: $R_1(X);\ W_2(X);\ C_2;\ R_1(X)$

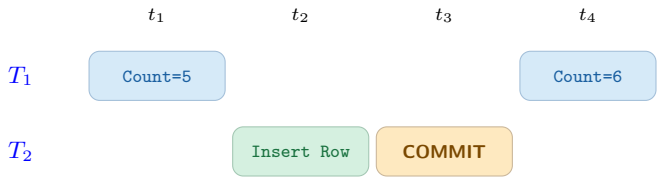


Problem: T_1 reads $X = 10$ first, then $X = 99$ — same transaction, same item, different values.

Example 339: Phantom Read — Worked Example

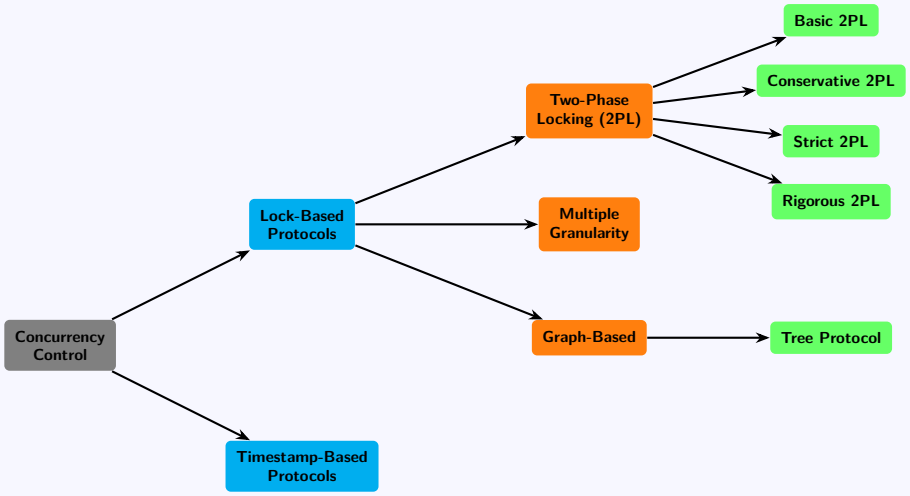
Scenario: Transaction T_1 counts employees with salary > 50000.

Schedule: $Q_1;\ I_2;\ C_2;\ Q_1$



Problem: The same query executed twice by T_1 returns different numbers of rows because T_2 inserted a new record. This newly appearing record is called a **phantom**.

Concurrency Control Protocol Hierarchy



8.2.1 Lock-Based Protocols

Lock-Based Protocols — Overview

A **lock-based protocol** requires a transaction to obtain a *lock* on a data item before accessing it. Locks ensure controlled, serialized access to shared data.

Basic rule:

Read $\Rightarrow$ acquire S-Lock Write $\Rightarrow$ acquire X-Lock

Types of locks:

- **Shared Lock (S-Lock):** Multiple transactions may hold S-locks on the same item simultaneously — all read, none write.
- **Exclusive Lock (X-Lock):** Only one transaction holds an X-lock; all other read and write requests are blocked.

Lock operations: `Lock-S(X)` `Lock-X(X)` `Unlock(X)`

Lock Compatibility Matrix

The lock manager consults this matrix before granting a requested lock.

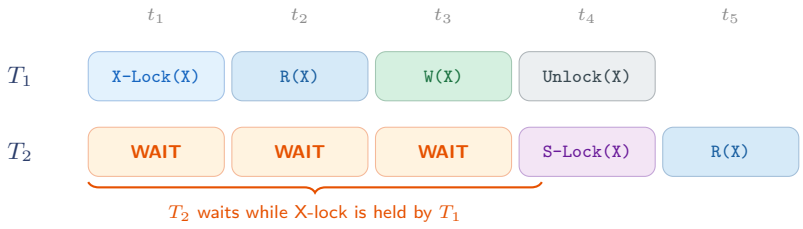
Current Lock ↓ Requested →	S (Shared)	X (Exclusive)
None	Compatible ✓	Compatible ✓
Shared (S)	Compatible ✓	Not Compatible ×
Exclusive (X)	Not Compatible ×	Not Compatible ×

- **No lock:** Both S and X can be granted immediately.
- **S-lock held:** More S-locks allowed; X-lock must wait (prevents dirty write).
- **X-lock held:** No other lock can be granted (full isolation).

Example 340: Locking Example with Two Transactions

Without locks: T_2 could read while T_1 is writing — dirty read.
With locks:

T_1 : X `R(X)` `W(X)` X T_2 : X `R(X)` X

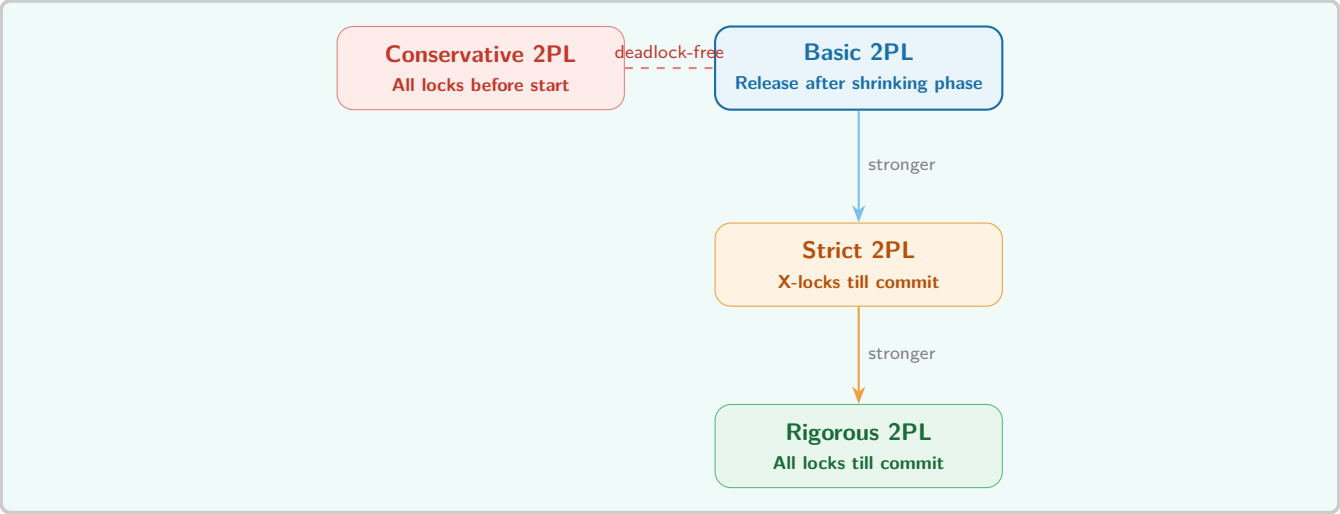


Result: T_2 cannot read while T_1 writes. Isolation is preserved.

8.2.1.1 Two-Phase Locking (2PL) Variants

Overview — 2PL Variants Hierarchy

All 2PL variants divide a transaction into at most two phases. They differ in *when* and *which* locks are released:



Basic Two-Phase Locking (Basic 2PL)

Basic Two-Phase Locking (2PL)

Two-Phase Locking (2PL) is a lock-based concurrency control protocol that guarantees *conflict-serializability*. Every transaction has exactly two phases:

1. **Growing Phase:** Locks may be acquired; *no* locks may be released.
2. **Shrinking Phase:** Locks may be released; *no* new locks may be acquired.

The transition point from growing to shrinking is called the **lock point**. Once a transaction releases its first lock it can never acquire another.

Basic 2PL — Protocol Rules

- A transaction must hold the appropriate lock (S or X) before any data access.
- A lock *upgrade* (S $\rightarrow$ X) is allowed during the growing phase if no other transaction holds an S-lock on the item.
- **Guarantees:** Conflict-serializability.
- **Does NOT prevent:** Deadlocks, cascading aborts, dirty reads (locks may be released before commit).

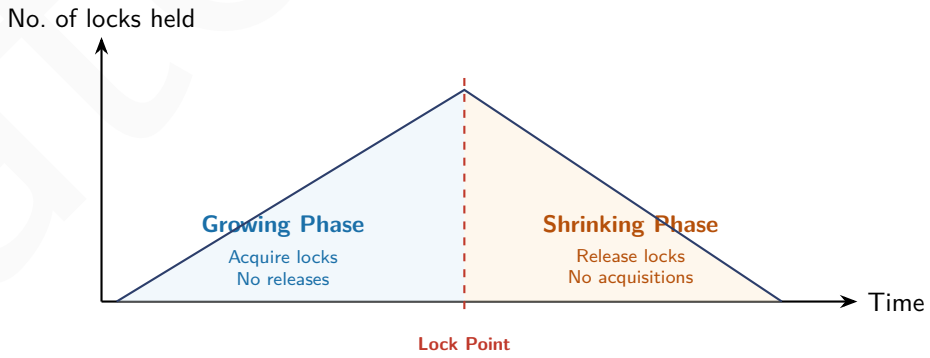


Figure 8.1: Basic 2PL lock acquisition profile — growing then shrinking

Example 341: Basic 2PL Execution Example

Transactions: $T_1 : R(A),\ W(A) \quad T_2 : R(A),\ W(A)$

Time	T_1	T_2
t_1	A (granted)	
t_2	`R(A)`	A (granted — S compatible)
t_3	Upgrade to A — waits for T_2 to release S	
t_4	(waiting)	A
t_5	A (granted)	A — waits for T_1's X
t_6	`W(A)`	(waiting)
t_7	A	A (granted)
t_8		`R(A)` , then A, `W(A)`

Note: Basic 2PL ensures serializability but S-locks were released before commit — cascading aborts are still possible.

Conservative Two-Phase Locking (Static 2PL)

Conservative Two-Phase Locking (Static 2PL)

Conservative 2PL (also called *Static 2PL*) eliminates deadlocks by requiring a transaction to obtain **all** necessary locks *before execution begins*.

If even one required lock is unavailable, the transaction acquires *no locks at all* and waits — partial lock holding never occurs.

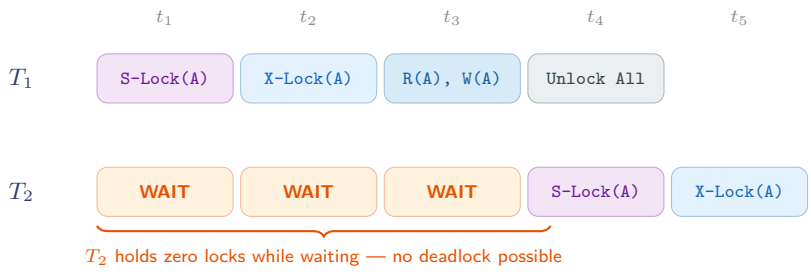
Conservative 2PL — Protocol Rules

- Request all required locks atomically at the start.
- If any lock is denied, wait and do **not** hold partial locks.
- Once all locks are granted, execute fully then release all.
- **Guarantees:** Conflict-serializability + **Deadlock-free**.
- **Drawback:** Requires complete foreknowledge of data items; reduces concurrency.

Example 342: Conservative 2PL Example

Transactions: $T_1 : R(A),\ W(A) \quad T_2 : R(A),\ W(A)$

Step-by-step:



Time	T_1	T_2
t_1	Request S(A) + X(A) — both granted	Request S(A) + X(A) — denied
t_2	Execute $R(A),\ W(A)$	Waiting (holds no locks)
t_3	Release all locks	
t_4		Request S(A) + X(A) — granted
t_5		Execute and release

Resulting order: $T_1 \to T_2$ (serial).

Strict Two-Phase Locking (Strict 2PL)

Strict Two-Phase Locking (Strict 2PL)

Strict 2PL adds a crucial restriction on top of Basic 2PL: *all exclusive (write) locks are held until the transaction commits or aborts.*
Shared locks may still be released early, but no uncommitted write is visible to any other transaction. This guarantees **cascadeless** and **recoverable** schedules in addition to conflict-serializability.

Strict 2PL — Protocol Rules & Guarantees

- Growing and shrinking phases as in Basic 2PL.
- **All X-locks are held until commit or abort** — never released during execution.
- S-locks may be released before commit (implementation-dependent).
- No other transaction can read or write data modified by an uncommitted transaction.

Guarantees:

✓ Conflict-serializability
✓ Recoverable schedules
✓ Cascadeless (ACA) schedules

Example 343: Strict 2PL Execution Example

Transactions: $T_1 : R(A),\ W(A) \quad T_2 : R(A),\ W(A)$

Time	T_1	T_2
t_1	A, R(A)	
t_2	A, W(A)	Requests A — blocked
t_3	(X-lock held till here)	Still blocked
t_4	A	A granted
t_5		R(A) , upgrade to A, W(A)

Key property: Because T_1's X-lock is held until commit, T_2 never sees an uncommitted write — cascading aborts are impossible.

Rigorous Two-Phase Locking (Rigorous 2PL)

Rigorous Two-Phase Locking (Rigorous 2PL)

Rigorous 2PL is the strictest 2PL variant. **All locks — both shared (S) and exclusive (X) — are held until the transaction commits or aborts.** No lock of any type is released before transaction completion.

As a direct consequence, the serialization order among transactions equals exactly their **commit order**.

Rigorous 2PL — Protocol Rules & Guarantees

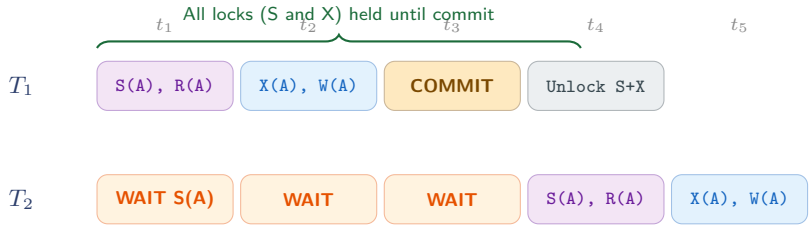
- Transactions acquire locks as needed (growing phase only).
- **No lock — S or X — is ever released before commit/abort.**
- Blocks all readers and writers until the holding transaction finishes.

Guarantees (strongest among 2PL):

✓ Conflict-serializability
✓ Recoverable schedules
✓ Cascadeless schedules
✓ Strict schedules
✓ Serialization order = commit order

Example 344: Rigorous 2PL Execution Example

Transactions: $T_1 : R(A),\ W(A)$ $\quad T_2 : R(A),\ W(A)$



Time	T_1	T_2
t_1	A, R(A)	Blocked on S(A)
t_2	A, W(A)	Still blocked
t_3		Still blocked (both S and X held)
t_4	Unlock all (S+X)	A granted
t_5		R(A) , A, W(A) ,

Distinction from Strict 2PL: In Strict 2PL, T_2 might have been able to obtain an S-lock after T_1's commit (same as here), but in theory S-locks in Strict 2PL can be released before commit. In Rigorous 2PL, *no lock ever is.*

Comparison of All 2PL Variants

Variant	Deadlock Free	Conflict Serial.	Recoverable	Cascadeless	Strict
Basic 2PL	×	✓	×	×	×
Conservative 2PL	✓	✓	×	×	×
Strict 2PL	×	✓	✓	✓	×
Rigorous 2PL	×	✓	✓	✓	✓

8.2.1.2 Multiple Granularity Locking

Multiple Granularity Locking — Overview

Multiple Granularity Locking (MGL) allows transactions to acquire locks at different *levels* of a data hierarchy:

Database → **Table** → **Page** → **Tuple**

A coarse-grain lock (e.g. table lock) is efficient but blocks many operations; a fine-grain lock (e.g. tuple lock) allows high concurrency but incurs lock-management overhead. MGL provides a *balanced* approach.

Intention locks (IS, IX, SIX) signal to the lock manager that a transaction *intends* to lock items at a lower level, enabling safe coexistence of locks at different levels.

Granularity Levels

- **Coarse Granularity:** Entire tables or files. Fewer locks, low overhead, low concurrency.
- **Medium Granularity:** Pages or blocks.
- **Fine Granularity:** Individual tuples or fields. Many locks, high overhead, high concurrency.

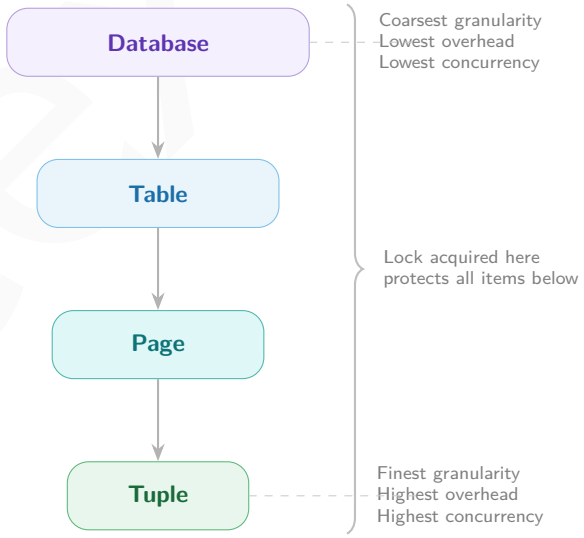


Figure 8.2: Four-level granularity hierarchy in MGL

Lock Modes in Multiple Granularity Locking

Mode	Symbol	Meaning
Shared	S	Read this node; no other transaction may write it.
Exclusive	X	Read/write this node; no other transaction may access it.
Intent-Shared	IS	Intends to place S-locks on *descendant* nodes.
Intent-Exclusive	IX	Intends to place X-locks (or S+X) on *descendant* nodes.
SIX	SIX	S-lock on this node *plus* intent to X-lock descendants.

IS $\Rightarrow$ will take S-locks below IX $\Rightarrow$ will take X-locks below

SIX $\Rightarrow$ S here + IX below

MGL Lock Compatibility Matrix

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	×
IX	✓	✓	×	×	×
S	✓	×	✓	×	×
SIX	✓	×	×	×	×
X	×	×	×	×	×

✓ = Compatible (both grants proceed) × = Not Compatible (newer request must wait)

Multiple Granularity Locking — Protocol Rules

1. Locks must be **acquired top-down** in the hierarchy (root $\rightarrow$ leaf direction).
2. Locks must be **released bottom-up** (leaf $\rightarrow$ root direction).
3. Before placing an **S-lock** on a node, the transaction must hold an **IS-lock** (or stronger) on its parent.
4. Before placing an **X-lock** on a node, the transaction must hold an **IX-lock** (or stronger) on its parent.
5. A node may be unlocked only after *all its children are unlocked*.

Example 345: Multiple Granularity Locking — Worked Example

Scenario: T_1 wants to update a single tuple in Table A. T_2 wants to read the entire Table A.

T_1 **(fine-grain write):**

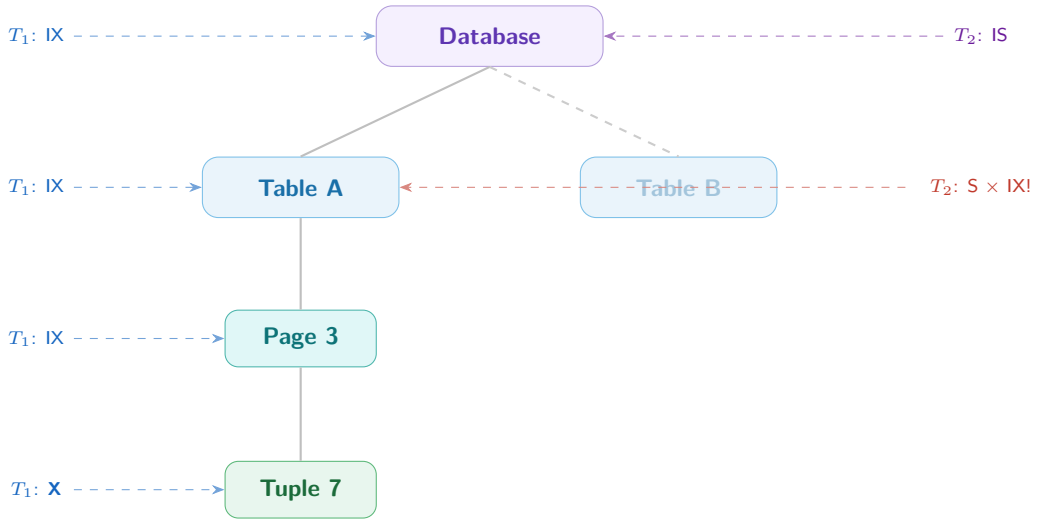
1. Lock **Database** with IX (intends X-lock below)
2. Lock **Table A** with IX (intends X-lock on a page)
3. Lock **Page 3** with IX (intends X-lock on a tuple)
4. Lock **Tuple 7** with X (exclusive write)

T_2 **(coarse-grain read):**

1. Lock **Database** with IS (intends S-lock below)

2. Lock **Table A** with S

Compatibility check: T_1 holds IX on Table A; T_2 requests S on Table A. From the matrix: IX $\times$ S = **Not Compatible** — T_2 must wait until T_1 releases IX on Table A.



Key benefit: T_2 does not have to lock individual tuples to check for conflicts — the IX-lock on the table already signals the conflict. This avoids checking millions of tuple-level locks.

8.2.1.3 Graph-Based Protocols

Graph-Based Protocols — Core Concept

Graph-based protocols enforce a *predefined partial order* on data items using a directed acyclic graph (DAG). A transaction may only lock data items in the order defined by the graph — from ancestor to descendant.
Since every transaction acquires locks in the same topological order, no circular wait can form and **deadlock is structurally impossible**.

Key contrast with 2PL:

- Graph protocols do *not* require two-phase behaviour.
- Locks may be released at any time (even before commit).
- The graph order substitutes for the growing/shrinking phase discipline.

Why Graph-Based Protocols are Used

- A DAG imposes ordering constraints — cycles in wait-for graph cannot occur.
- Guarantees conflict-serializable schedules.
- **Deadlock-free by design.**
- Suitable when data items have a natural hierarchical or dependency order (e.g. B-tree indices, filesystem hierarchies).
- Allows **early unlocking** — higher concurrency than strict 2PL.

Graph Representation

- Each data item is a **node** in a directed graph.
- A directed edge $Q_i \to Q_j$ means: any transaction that locks Q_j *must already hold* a lock on Q_i.
- The graph must be **acyclic** (a DAG).
- All transactions must obey the graph order when acquiring locks.

$Q_1 \to Q_2 \to Q_3 \quad \Rightarrow \quad$ a transaction must lock Q_1 before Q_2 and Q_2 before Q_3

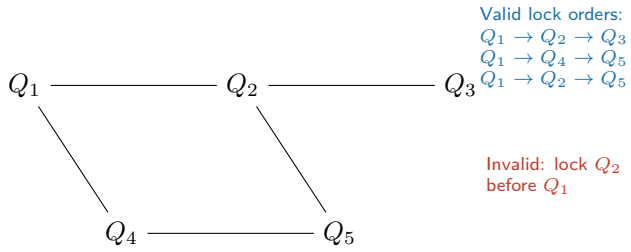


Figure 8.3: Example DAG — lock order must follow edge direction

Graph-Based Locking Rules

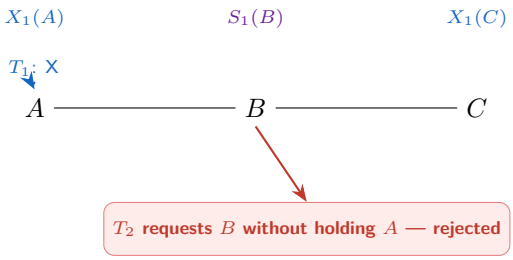
1. A transaction's **first lock** may be placed on any node.
2. Thereafter, a node Q_j may be locked only if the transaction currently holds a lock on Q_j's **parent** in the graph.
3. Locks may be **released at any time** — no shrinking-phase constraint.
4. Once a node is unlocked, it **cannot be relocked** by the same transaction.
5. Both S-locks and X-locks are used as usual.

Properties of Graph-Based Protocols

- **Conflict-serializable** — topological order ensures no cycle in the serialization graph.
- **Deadlock-free** — the acyclic lock-order prevents circular waiting.
- **No 2PL required** — locks may be released before all locks are acquired.
- **Early unlock** — higher concurrency than strict 2PL.
- **Not necessarily cascadeless** — dirty reads can still occur since X-locks may be released before commit.

Example 346: title

Data item graph: $A \to B \to C$
Transactions:



Step	T_1	T_2
1	$X_1(A)$ — granted (first lock)	Tries to lock B directly
2	$S_1(B)$ — granted (holds A)	Rejected — must hold A first
3	Unlock A	Requests $X_2(A)$ — granted
4	$X_1(C)$ — granted (holds B)	$S_2(B)$ — granted (holds A)
5	Unlock B, C	Unlock A, B

Result: No cyclic wait occurs because both transactions follow the same $A \to B \to C$ order.

8.2.1.4 Tree-Based Protocols

Tree-Based Protocols — Core Concept

Tree-based protocols are a special case of graph-based protocols where the data items form a *rooted tree*. Lock acquisition must follow the tree structure strictly from parent to child.
The tree protocol guarantees **conflict-serializability** and is **deadlock-free** without requiring two-phase locking, and allows **early unlocking** for higher concurrency.

Why Tree-Based Protocols are Used

- **Natural for hierarchical data:** B-trees, file systems, XML documents, organisational hierarchies.
- **Deadlock-free** — top-down ordering prevents cyclic waits.
- **Early unlock allowed** — parent can be released as soon as child is locked, unlike strict 2PL.
- **Higher concurrency** than 2PL variants that hold locks until commit.
- Guarantees conflict-serializable schedules.

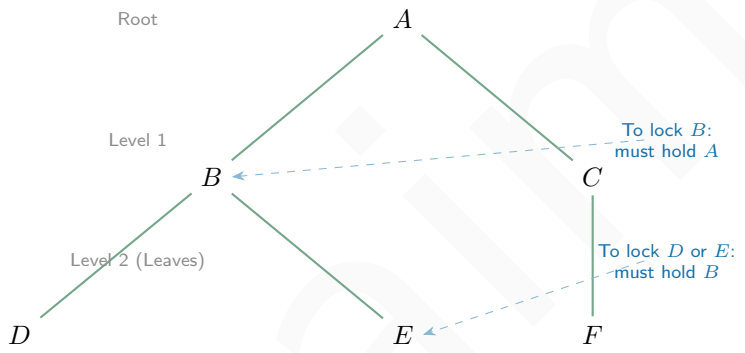


Figure 8.4: Tree protocol — locks must be acquired top-down (parent before child)

Tree Representation

- Data items form a **rooted tree**.
- A **parent–child** edge defines the locking order.
- A node may be locked only if its parent is *currently locked* by the same transaction.
- The **root** has no parent and may be locked at any time.
- Structure:

$$\text{Root} \rightarrow \text{Internal Nodes} \rightarrow \text{Leaves}$$

Tree Protocol — Locking Rules

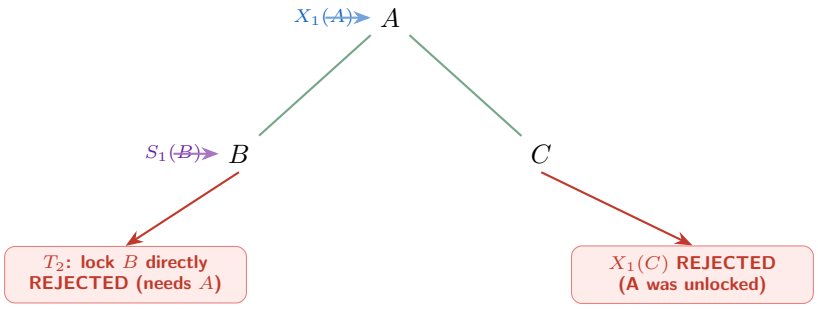
1. The **first lock** of a transaction may be on *any* node (not necessarily the root).
2. Any subsequent lock on node Q requires that the transaction **currently holds** the lock on Q's parent.
3. Locks may be **released at any time**.
4. Once a lock on a node is released, that node **cannot be locked again** by the same transaction.
5. Both S-locks (read) and X-locks (write) are used.

Properties of the Tree Protocol

Property	Holds?	Explanation
Conflict-serializable	✓	Top-down order prevents serialization cycles
Deadlock-free	✓	Acyclic order — no circular waiting possible
Cascadeless	×	Dirty reads possible (locks released before commit)
Two-phase locking	×	Not required — locks may be released at any time
Early unlock	✓	Higher concurrency than basic 2PL

Example 347: title

Tree structure: $A \to \{B, C\}$ (A is parent of both B and C)



Step	T_1	T_2
1	$X_1(A)$ — granted (first lock)	Tries $X_2(B)$ directly
2	$S_1(B)$ — granted (holds A)	Rejected — must lock A first
3	Unlock A	$X_2(A)$ — granted
4	$X_1(C)$ — REJECTED (A unlocked!)	$S_2(B)$ — granted (holds A)
5	Unlock B	Unlock A, B

[title=Key Restriction to Remember] Once a transaction unlocks a node, it **cannot re-lock it** and cannot lock any ancestor that it no longer holds. In Step 4 above, T_1 unlocked A before locking C, so $X_1(C)$ is rejected — the path $A \to C$ is broken. This rule prevents re-locking cycles while preserving the early unlock advantage.

8.2.2 Timestamp-Based Protocols

Timestamp-Based Protocols — Core Concept

Timestamp-based protocols assign each transaction T_i a unique timestamp $TS(T_i)$ at start time. Rather than using locks, the protocol *orders conflicting operations in timestamp order* to guarantee serializability.
Each data item Q maintains two counters:

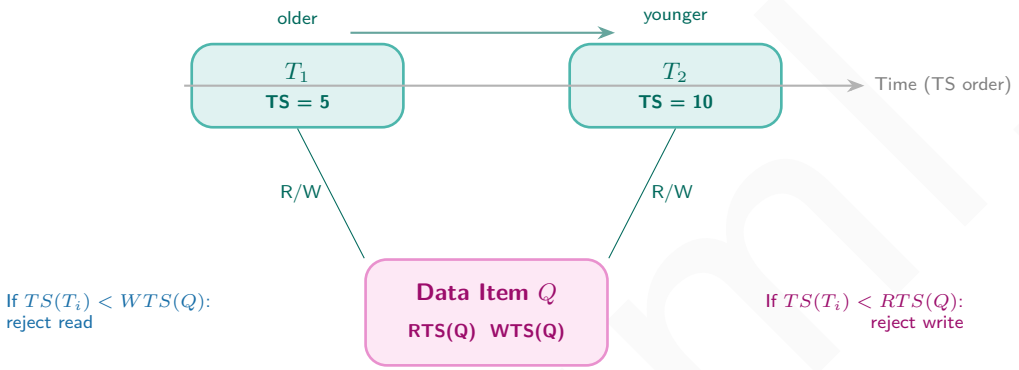
$$\underbrace{RTS(Q)}_{\text{largest TS of any reader of } Q} \qquad \underbrace{WTS(Q)}_{\text{largest TS of any writer of } Q}$$

If an operation would violate timestamp order, the transaction is **rolled back and restarted** with a new (larger)

timestamp rather than being made to wait.

Why Timestamp Ordering is Needed

- Provides concurrency control **without locks** — no lock-acquisition overhead.
- Guarantees **conflict-serializable** schedules by enforcing a global timestamp order.
- **Deadlock-free**: transactions abort immediately instead of waiting cyclically.
- Useful in high-contention environments where lock waiting is costly.
- Maintains a serial order equal to the timestamp assignment order.



8.2.2.1 Basic Timestamp Ordering

Basic Timestamp Ordering — Read & Write Rules

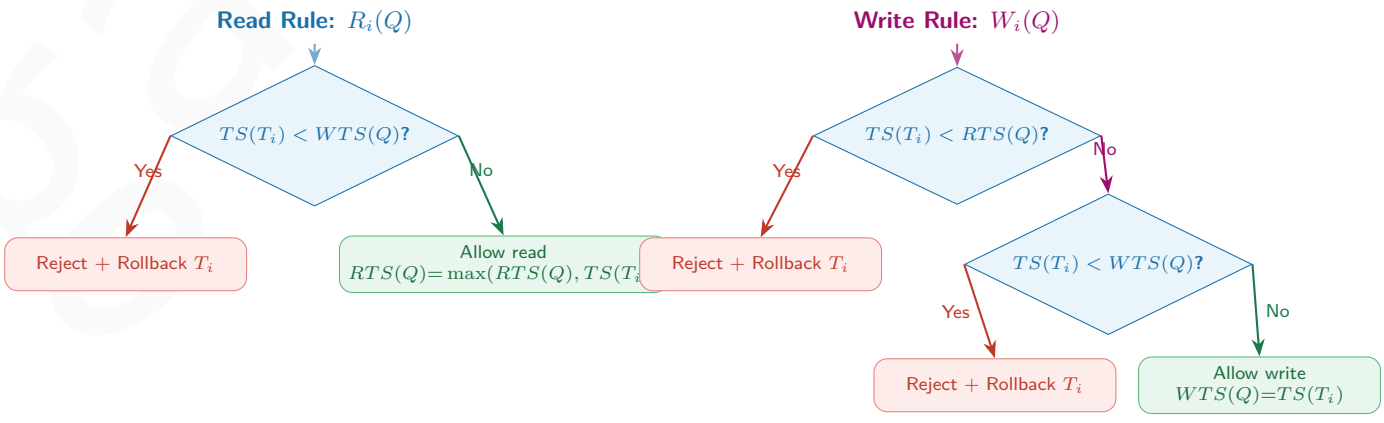
Let transaction T_i with timestamp $TS(T_i)$ operate on data item Q.

Read Rule $[R_i(Q)]$:

- If $TS(T_i) < WTS(Q) \Rightarrow$ **reject** and rollback T_i (a newer transaction already wrote — reading would violate order)
- Otherwise **allow** and update: $RTS(Q) = \max\big(RTS(Q),\, TS(T_i)\big)$

Write Rule $[W_i(Q)]$:

- If $TS(T_i) < RTS(Q) \Rightarrow$ **reject** and rollback T_i (a newer transaction already read — overwriting would violate order)
- If $TS(T_i) < WTS(Q) \Rightarrow$ **reject** and rollback T_i (a newer transaction already wrote — obsolete write)
- Otherwise **allow** and set: $WTS(Q) = TS(T_i)$



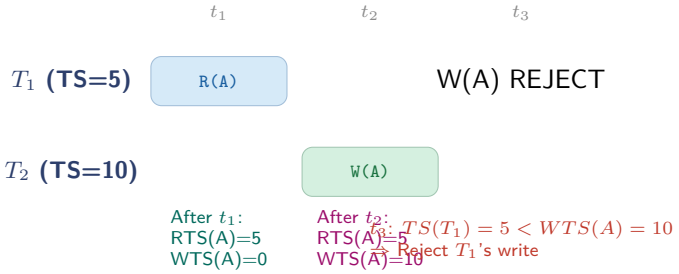
Properties of Timestamp Ordering

- **Conflict-serializable:** Enforces timestamp order globally.
- **Deadlock-free:** Transactions abort immediately — no waiting.
- **More rollbacks:** Compared to locking; aborted transactions restart with a new TS.
- **Priority:** Older (lower-TS) transactions are preferred.
- **Starvation risk:** A transaction may repeatedly abort if newer transactions keep accessing the same items.

Example 348: title

Given:

$$TS(T_1) = 5, \quad TS(T_2) = 10 \qquad \text{Initial: } RTS(A) = 0,\ WTS(A) = 0$$



Step	Operation	Check	Result	RTS(A)	WTS(A)
t_1	$R_1(A)$	$TS(T_1) = 5 \geq WTS(A) = 0$	✓ Allow	5	0
t_2	$W_2(A)$	$TS(T_2) = 10 \geq RTS(A) = 5$ & $\geq WTS(A) = 0$	✓ Allow	5	10
t_3	$W_1(A)$	$TS(T_1) = 5 < WTS(A) = 10$	× Rollback T_1	5	10

Interpretation: T_1 is older (TS=5) but tries to write after T_2 (TS=10) already wrote — this would violate the serial order $T_1 \to T_2$. The protocol correctly rejects T_1's write.

8.2.2.2 Thomas Write Rule — Optimization

Thomas Write Rule — Optimization

The **Thomas Write Rule** is an optimization of the basic timestamp write rule. It avoids unnecessary rollbacks for *obsolete writes*.

Modified Write Rule for $W_i(Q)$:

1. If $TS(T_i) < RTS(Q)$ ⇒ **rollback** T_i (a newer read exists).
2. If $TS(T_i) < WTS(Q)$ ⇒ **ignore** the write (it is obsolete — a newer write already exists). Do *not* abort.
3. Otherwise **allow** and set $WTS(Q) = TS(T_i)$.

Condition	Basic TSO	Thomas Write Rule
$TS(T_i) < RTS(Q)$	Rollback	Rollback
$TS(T_i) < WTS(Q)$	Rollback	Ignore (skip write)
Otherwise	Allow	Allow

Effect: Reduces unnecessary rollbacks while still guaranteeing serializable (specifically view-serializable) schedules.

Example 349: Thomas Write Rule — Comparison with Basic TSO

Given: $TS(T_1) = 5$, $TS(T_2) = 10$, $WTS(A) = 10$ (after T_2 wrote).

T_1 **now requests** $W_1(A)$**:**

Protocol	Check	Action
Basic TSO	$TS(T_1) = 5 < WTS(A) = 10$	Rollback T_1
Thomas Write Rule	$TS(T_1) = 5 < WTS(A) = 10$	Ignore write; T_1 continues

Benefit: T_1's write would have been overwritten by T_2's write anyway — it is *obsolete*. Ignoring it avoids an unnecessary rollback with no loss of correctness.

8.2.2.3 Strict Timestamp Ordering Protocol

Strict Timestamp Ordering (STO)

Strict Timestamp Ordering is a refinement of the basic timestamp ordering protocol that prevents cascading aborts by enforcing the following rule:

- If a transaction writes a data item, other transactions cannot read or write that item until the writing transaction either **commits or aborts**.
- This ensures that no transaction ever reads uncommitted data.

Hence Strict TSO guarantees:

- Conflict Serializability
- Recoverable schedules
- Cascadeless schedules

However, unlike Basic Timestamp Ordering, transactions may **wait** for other transactions.

Read Rule: $R_i(Q)$

When transaction T_i issues a read operation $R_i(Q)$:

- If $TS(T_i) < WTS(Q)$

  $$\Rightarrow \textbf{Rollback } T_i$$

  because the transaction is trying to read a value written by a **newer transaction**.
- If $TS(T_i) > WTS(Q)$ and the last writer of Q has **not committed**

  $$\Rightarrow \textbf{Wait}$$

  because Strict TSO does not allow reading uncommitted values.
- Otherwise

  $$\Rightarrow \textbf{Allow read}$$

  and update

  $$RTS(Q) = \max(RTS(Q), TS(T_i))$$

Example 350: Read Rule Case 1: Waiting Situation

Consider the following timestamps:

$$TS(T_1) = 10, \qquad TS(T_2) = 20$$

Transaction schedule:

$$W_1(Q) \quad R_2(Q)$$

Step-by-step analysis:

1. $W_1(Q)$ executes.

$$WTS(Q) = 10$$

2. T_2 attempts $R_2(Q)$.
   Since

$$TS(T_2) = 20 > WTS(Q) = 10$$

   normally the read would be allowed.
   However the writer T_1 has **not yet committed**.

Therefore according to Strict TSO:

$$R_2(Q) \Rightarrow \textbf{WAIT}$$

T_2 must wait until T_1 commits.

Example 351: Read Rule Case 2: Allowed

Consider the schedule

$$W_1(Q),\ Commit_1,\ R_2(Q)$$

with

$$TS(T_1) = 10, \qquad TS(T_2) = 20$$

Step-by-step:

1. $W_1(Q)$ executes

$$WTS(Q) = 10$$

2. T_1 commits.
3. T_2 executes $R_2(Q)$.
   Since

$$TS(T_2) = 20 > WTS(Q) = 10$$

   and the previous writer has already committed, the read is allowed.

Update:

$$RTS(Q) = 20$$

Thus the operation is **allowed**.

Write Rule: $W_i(Q)$

When transaction T_i issues a write operation $W_i(Q)$:

- If

$$TS(T_i) < RTS(Q)$$

or

$$TS(T_i) < WTS(Q)$$

then

$$\Rightarrow \textbf{Rollback } T_i$$

because a younger transaction has already read or written the item.

- If

$$TS(T_i) > WTS(Q)$$

but the last writer has **not committed**

$$\Rightarrow \textbf{Wait}$$

because Strict TSO prevents overwriting uncommitted values.

- Otherwise

$$\Rightarrow \textbf{Allow write}$$

and update

$$WTS(Q) = TS(T_i)$$

Example 352: Write Rule Case 1: Waiting Situation

Let

$$TS(T_1) = 10, \qquad TS(T_2) = 20$$

Schedule:

$$W_1(Q),\ W_2(Q)$$

Execution:

1. $W_1(Q)$ executes.

$$WTS(Q) = 10$$

2. T_2 attempts $W_2(Q)$.
   Since

$$TS(T_2) = 20 > WTS(Q) = 10$$

normally the write would be allowed.

However T_1 has **not committed yet**.

Therefore

$$W_2(Q) \Rightarrow \textbf{WAIT}$$

until T_1 commits.

Example 353: Write Rule Case 2: Allowed

Schedule:

$$W_1(Q),\ Commit_1,\ W_2(Q)$$

with

$$TS(T_1) = 10, \qquad TS(T_2) = 20$$

Step-by-step:

1. $W_1(Q)$ executes

   $$WTS(Q) = 10$$

2. T_1 commits.
3. T_2 executes $W_2(Q)$.

   Since

   $$TS(T_2) = 20 > WTS(Q) = 10$$

   and the previous writer has committed, the write is allowed.

Update:

$$WTS(Q) = 20$$

Properties of Strict TSO

- **Conflict Serializable**

  All operations respect the timestamp order of transactions.
- **Recoverable**

  A transaction only reads committed data.
- **Cascadeless**

  If a transaction aborts, other transactions are not forced to abort.
- **Deadlocks Possible**

  Unlike Basic TSO, transactions may wait for others, which can produce circular waiting.
- **Lower Concurrency**

  Waiting for commits can delay execution.

Example 354: Schedule Analysis using Basic TSO

Given schedule

$$S: W_1(Q),\ R_2(Q),\ W_2(P),\ R_3(P),\ W_3(P),\ W_3(Q)$$

with timestamps

$$TS(T_1) = 10, \quad TS(T_2) = 20, \quad TS(T_3) = 30$$

Initial values:

$$RTS(Q) = 0, \quad WTS(Q) = 0$$

$$RTS(P) = 0, \quad WTS(P) = 0$$

Step 1: $W_1(Q)$

$$TS(T_1) = 10 > RTS(Q) = 0$$

Write allowed.

$$WTS(Q) = 10$$

Step 2: $R_2(Q)$

$$TS(T_2) = 20 > WTS(Q) = 10$$

Read allowed.

$$RTS(Q) = 20$$

Step 3: $W_2(P)$

$$TS(T_2) = 20 > RTS(P) = 0$$

Write allowed.

$$WTS(P) = 20$$

Step 4: $R_3(P)$

$$TS(T_3) = 30 > WTS(P) = 20$$

Read allowed.

$$RTS(P) = 30$$

Step 5: $W_3(P)$
Check:

$$TS(T_3) = 30 > RTS(P) = 30$$

Allowed.

$$WTS(P) = 30$$

Step 6: $W_3(Q)$
Check

$$TS(T_3) = 30 > RTS(Q) = 20$$

Allowed.

$$WTS(Q) = 30$$

Thus the entire schedule executes successfully.

Example 355: Schedule Analysis using Strict TSO

Schedule:

$$S: W_1(Q),\ R_2(Q),\ W_2(P),\ R_3(P),\ W_3(P),\ W_3(Q),\ Commit_1,\ Commit_2$$

Timestamps:

$$TS(T_1) = 10, \quad TS(T_2) = 20, \quad TS(T_3) = 30$$

Step 1: $W_1(Q)$
Write allowed.

$$WTS(Q) = 10$$

Step 2: $R_2(Q)$
T_2 attempts to read Q.
But T_1 has **not committed yet**.
Therefore

$$R_2(Q) \Rightarrow WAIT$$

Step 3: $Commit_1$
Now the write becomes committed.
Step 4: Resume $R_2(Q)$
Read allowed.

$$RTS(Q) = 20$$

The remaining operations proceed similarly respecting the timestamp order.

Comparison — All Non-Lock Protocols

Protocol	Deadlock-Free	Conflict Serial.	Cascadeless	Early Unlock	Uses Locks
Basic TSO	✓	✓	×	N/A	×
Strict TSO	×	✓	✓	N/A	×
Thomas Write	✓	✓*	×	N/A	×
Graph-Based	✓	✓	×	✓	✓
Tree-Based	✓	✓	×	✓	✓

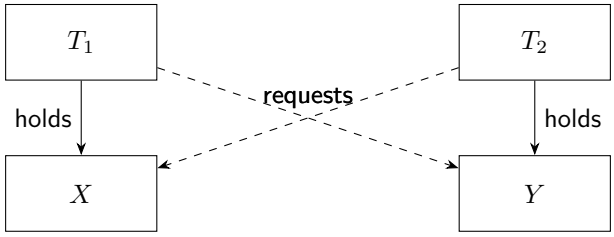
*Thomas Write Rule guarantees view-serializability (slightly weaker than conflict-serializability).

8.3 Deadlock Handling in DBMS

Deadlock Concept

A **deadlock** occurs when two or more transactions are waiting indefinitely for resources locked by each other.
In locking-based concurrency control, deadlocks arise when transactions hold locks and request additional locks held by others.

Example 356: Example of Deadlock



Here:

- T_1 holds X and requests Y
- T_2 holds Y and requests X

Both wait forever ⇒ **Deadlock**.

Necessary Conditions for Deadlock

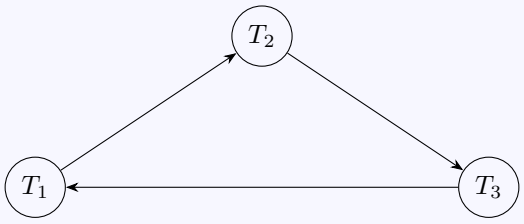
Deadlock can occur only if the following four conditions hold (**Coffman conditions**):

1. **Mutual Exclusion**
   Only one transaction can hold a resource at a time.
2. **Hold and Wait**
   A transaction holding a resource requests another resource.
3. **No Preemption**
   Resources cannot be forcibly taken away.
4. **Circular Wait**
   Transactions form a cycle of waiting.

Wait-For Graph (Deadlock Detection)

A **Wait-For Graph (WFG)** is used to detect deadlocks.

- Nodes represent transactions
- Edge $T_i \rightarrow T_j$ means T_i waits for T_j



Cycle in Wait-For Graph ⇒ Deadlock

8.3.1 Deadlock Handling Techniques

DBMS can handle deadlocks using three approaches:

1. **Deadlock Prevention**
2. **Deadlock Detection and Recovery**

3. **Deadlock Avoidance**

Deadlock Prevention

Deadlock prevention ensures that at least one of the Coffman conditions never holds.
Two common timestamp-based techniques:

- Wait–Die
- Wound–Wait

8.3.1.1 Wound–Wait Scheme

Wound–Wait Rule

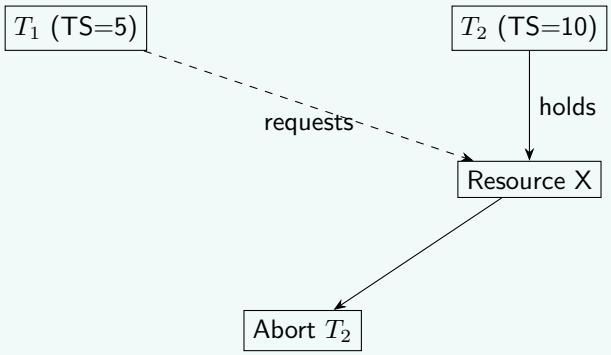
This protocol uses transaction timestamps.

$$TS(T_i) < TS(T_j) \Rightarrow T_i \text{ is older}$$

Case	Action	Explanation
Older requests lock from younger	Younger aborted	Older wounds younger
Younger requests lock from older	Younger waits	Older keeps resource

Key rule:

Older wounds younger



Since T_1 is older:

$$T_1 \text{ wounds } T_2$$

T_2 is aborted and T_1 gets the lock.

8.3.1.2 Wait–Die Scheme

Wait–Die Rule

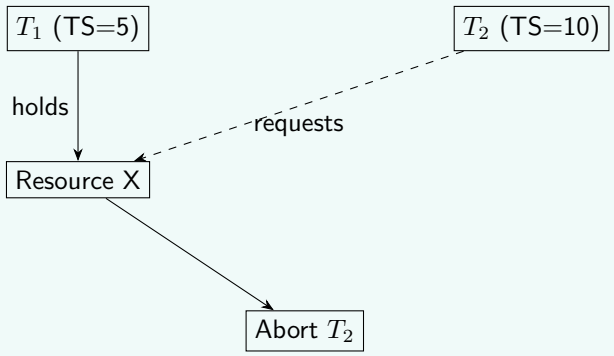
Another timestamp-based prevention protocol.

$$TS(T_i) < TS(T_j) \Rightarrow T_i \text{ is older}$$

Case	Action	Explanation
Older requests lock from younger	Wait	Older allowed to wait
Younger requests lock from older	Abort	Younger dies

Key rule:

Older waits, Younger dies



Since T_2 is younger:

T_2 dies (aborted)

Comparison: Wait–Die vs Wound–Wait

Feature	Wait–Die	Wound–Wait
Older transaction	Waits	Preempts younger
Younger transaction	Aborted	Waits
Preemption	No	Yes
Deadlock	Impossible	Impossible

8.4 Database Recovery Management

Goal of Database Recovery

The **Recovery Manager** ensures that the database preserves the ACID properties even when failures occur. To achieve this, DBMS maintains a **Log File** stored on **stable storage**. The log records every modification performed by transactions so that operations can be **undone** or **redone** after a crash.

Types of Failures

- **Transaction Failure**
  - Logical error
  - Integrity constraint violation
  - Explicit abort
- **System Crash**
  - Power failure
  - Operating system crash
  - Hardware malfunction
- **Disk Failure**
  - Head crash

– Media corruption

Log-based recovery mainly addresses **transaction failures** and **system crashes**. Disk failures require database **backups**.

8.4.1 Structure of Log Records

Log Format

The log file is a sequential append-only file.

- $\langle T_i, start \rangle$ Transaction T_i begins execution.
- $\langle T_i, X, V_{old}, V_{new} \rangle$ Transaction T_i changes data item X Old value $= V_{old}$, New value $= V_{new}$.
- $\langle T_i, commit \rangle$ Transaction successfully completes.
- $\langle T_i, abort \rangle$ Transaction is rolled back.
- $\langle checkpoint \rangle$ Indicates a recovery starting point.

Write Ahead Logging

Rule:

Log record must reach stable storage before the database page is written.

This rule guarantees that recovery information always exists if the system crashes.

8.4.2 Deferred Update vs Immediate Update

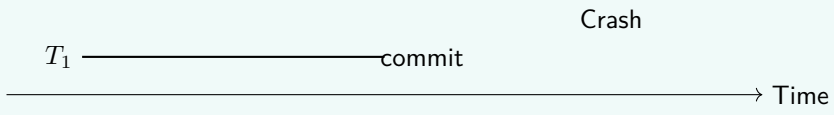
8.4.2.1 Deferred Update (NO-UNDO / REDO)

Deferred Update

- Database is updated only after commit.
- Updates are first stored in the log.
- If crash occurs before commit, nothing was written to the database.

Recovery Requirement:

Only **REDO** operations are required.



8.4.2.2 Immediate Update (UNDO / REDO)

Immediate Update

Database pages may be written before commit.
Therefore:

- Some committed transactions may need **REDO**.
- Some uncommitted transactions may require **UNDO**.

Both UNDO and REDO are required

8.4.2.3 Recovery Algorithm (UNDO / REDO)

Recovery Algorithm

After a crash, the recovery manager performs the following steps.
Step 1: Initialization

$$UndoList = \{\text{Transactions active at checkpoint}\}$$

$$RedoList = \emptyset$$

Step 2: Forward Log Scan
Scan the log from checkpoint to the end.

- If $\langle T_i, start \rangle$ appears Add T_i to $UndoList$.
- If $\langle T_i, commit \rangle$ appears Move T_i from $UndoList$ to $RedoList$.

Step 3: Recovery Actions

- Perform **UNDO** for transactions in $UndoList$ in reverse order.
- Perform **REDO** for transactions in $RedoList$ in forward order.

Example 357:

Consider the following log.

1	$\langle T_1, start \rangle$
2	$\langle T_1, A, 100, 200 \rangle$
3	$\langle T_1, commit \rangle$
4	$\langle T_2, start \rangle$
5	$\langle checkpoint(T_2) \rangle$
6	$\langle T_3, start \rangle$
7	$\langle T_2, B, 50, 100 \rangle$
8	$\langle T_3, C, 10, 20 \rangle$
9	$\langle T_2, commit \rangle$
10	$\langle T_4, start \rangle$
11	$\langle T_4, D, 0, 5 \rangle$
12	Crash

Step 1: Initialization

$$UndoList = \{T_2\}$$

$$RedoList = \emptyset$$

Step 2: Forward Scan

- T_3 starts UndoList $= \{T_2, T_3\}$
- T_2 commits RedoList $= \{T_2\}$ UndoList $= \{T_3\}$
- T_4 starts UndoList $= \{T_3, T_4\}$

Final Sets

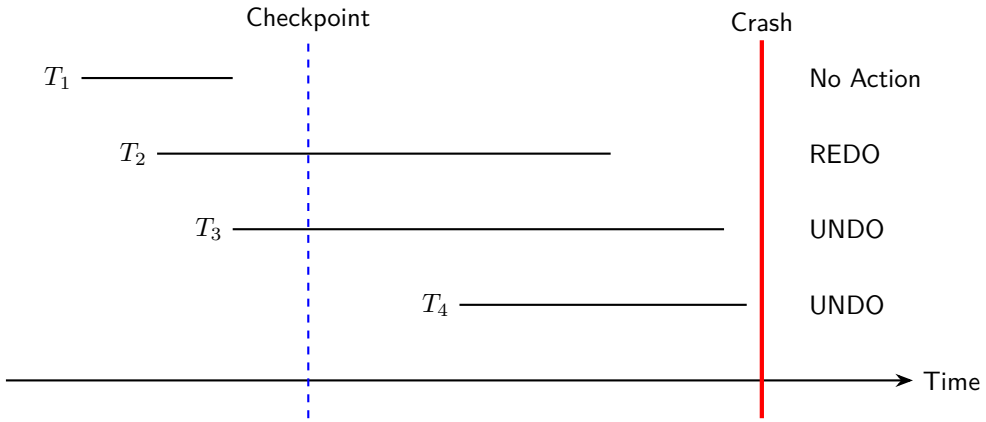
$$RedoList = \{T_2\}$$

$$UndoList = \{T_3, T_4\}$$

Recovery Actions

- Undo T_4
- Undo T_3
- Redo T_2

Checkpoint Timeline Visualization



8.5 Practice Problems

Problem 261 ***[MCQ]*** *A transaction T has executed its* last *statement and written all modified data to the* ***buffer cache****, but none of it has been flushed to stable storage yet. In which state does the standard state diagram place T?*
A. Active
B. Partially Committed
C. Committed
D. Failed

Problem 262 ***[MSQ]*** *Which of the following statements about the* ***Failed*** *state of a transaction are correct?*
A. A transaction can reach the Failed state from the Partially Committed state.
B. A transaction in the Failed state may be restarted as a new transaction after reaching Aborted.
C. The Failed state is a terminal state; no further transitions are defined from it.
D. A logical error (e.g., division by zero) inside the transaction can cause it to enter the Failed state.

Problem 263 ***[MSQ]*** *A banking transaction T_1 transfers \$500 from account A to account B. It reads A, deducts \$500, writes the updated A to disk, and then the system crashes* before *writing B. After recovery, account A has been debited but account B has not been credited. Which ACID property is* ***primarily*** *violated?*
A. Consistency
B. Isolation
C. Atomicity
D. Durability

Problem 264 ***[MCQ]*** *Transaction T_1 reads $X = 100$. Concurrently, T_2 updates X to 200 and commits. T_1 then reads X again within the same transaction and obtains the value 200. This anomaly is called:*
A. Dirty Read, violating Atomicity
B. Non-Repeatable Read, violating Isolation
C. Phantom Read, violating Consistency
D. Dirty Read, violating Isolation

Problem 265 ***[MCQ]*** *An integrity constraint states "account balance ≥ 0". Transaction T first deducts \$1000 from account A (making it $-\$200$), then immediately deposits \$1000 into account B, and finally commits. At no point does any other transaction observe the intermediate state. Which statement is* ***correct****?*
A. Consistency is violated because the constraint was broken mid-transaction.
B. Consistency is not violated; constraints need only hold at transaction boundaries.
C. Atomicity is violated because two accounts were modified.
D. Isolation is violated because the intermediate state existed, even if unobserved.

Problem 266 ***[MCQ]*** *Transaction T_1 has m operations and transaction T_2 has n operations. Assuming all orderings of individual operations are possible, the number of distinct interleaved schedules of T_1 and T_2 (preserving the internal order within each transaction) is:*
A. $m \times n$
B. $(m+n)!$
C. $\binom{m+n}{m}$
D. 2^{m+n}

Problem 267 ***[MCQ]*** *Which of the following correctly describes the relationship between* ***conflict serializability*** *and* ***view serializability****?*
A. Every view serializable schedule is also conflict serializable.
B. Every conflict serializable schedule is also view serializable.
C. The two notions coincide exactly when the schedule contains no blind writes.
D. Both B and C are correct.

Problem 268 ***[MSQ]*** *Which of the following statements about schedules involving n transactions are **correct**?*
A. There are exactly $n!$ possible serial schedules over n transactions.
B. Every serial schedule preserves database consistency, assuming each transaction individually preserves consistency.
C. Checking conflict serializability can be done in polynomial time in the size of the schedule.
D. Checking view serializability is NP-Hard.

Problem 269 ***[MCQ]*** *A schedule S over transactions T_1 and T_2 is given:*

$$S: \ R_1(A),\ W_2(A),\ R_2(B),\ W_1(B)$$

*Which of the following is **true** about S?*
A. S is conflict serializable and equivalent to $T_1 \to T_2$.
B. S is conflict serializable and equivalent to $T_2 \to T_1$.
*C. S is **not** conflict serializable because the precedence graph has a cycle.*
D. S is conflict serializable and equivalent to both $T_1 \to T_2$ and $T_2 \to T_1$.

Problem 270 ***[NAT]*** *Consider the schedule:*

$$S: R_1(X),\ W_2(X),\ R_2(Y),\ W_1(Y),\ W_3(X),\ W_3(Y)$$

*How many **conflicting pairs** of operations exist in S?*

Problem 271 ***[MCQ]*** *The precedence graph of a schedule S over transactions T_1, T_2, T_3 contains* exactly *the edges $T_1 \to T_2$, $T_2 \to T_3$, and $T_1 \to T_3$. Which conclusion is correct?*
A. S is not conflict serializable because a node has more than one outgoing edge.
B. S is conflict serializable and is equivalent to the serial order $T_1 \to T_2 \to T_3$ only.
C. S is conflict serializable and is equivalent to more than one serial order.
D. S is conflict serializable and is equivalent to $T_1 \to T_3 \to T_2$ only.

Problem 272 ***[MSQ]*** *Consider the schedule:*

$$S: W_1(X),\ R_2(X),\ W_2(Y),\ R_1(Y),\ C_1,\ C_2$$

*Which of the following are **true** about S?*
A. The precedence graph contains the edge $T_1 \to T_2$ (due to $W_1(X)$ before $R_2(X)$).
B. The precedence graph contains the edge $T_2 \to T_1$ (due to $W_2(Y)$ before $R_1(Y)$).
C. S is conflict serializable.
D. S is view serializable.

Problem 273 ***[MCQ]*** *A schedule S is view serializable but **not** conflict serializable. Which of the following must be true about S?*
A. S contains a dirty read.
B. S contains at least one blind write.
C. S has an acyclic precedence graph.
D. Both B and C.

Problem 274 ***[MCQ]*** *Consider the following schedule:*

$$S: W_1(X),\ W_2(X),\ W_3(X),\ C_1,\ C_2,\ C_3$$

*Which serial schedule(s) is/are **view equivalent** to S?*
A. $T_1 \to T_2 \to T_3$ only.
B. $T_2 \to T_1 \to T_3$ only.
C. Any serial order in which T_3 executes last.
D. All $3! = 6$ serial schedules.

Problem 275 ***[NAT]*** *Schedule S below contains only write operations and is **view serializable** but **not** conflict serializable:*

$$S : W_1(X),\ W_2(X),\ W_1(Y),\ W_2(Y),\ W_3(X),\ W_3(Y),\ C_1,\ C_2,\ C_3$$

*How many serial schedules over $\{T_1, T_2, T_3\}$ are **view equivalent** to S?*

Problem 276 ***[MCQ]*** *Which of the following schedules is **view serializable** but **NOT conflict serializable**?*
A. $R_1(X),\ W_2(X),\ R_2(Y),\ W_1(Y),\ C_1,\ C_2$
B. $W_1(X),\ W_2(X),\ W_2(Y),\ W_1(Y),\ W_3(X)$
C. $R_1(X),\ W_1(X),\ R_2(X),\ W_2(X),\ C_1,\ C_2$
D. $W_1(X),\ R_2(X),\ W_2(X),\ C_1,\ C_2$

Problem 277 ***[MCQ]*** *Consider the schedule:*

$$S : R_1(X),\ W_1(X),\ R_2(X),\ \textbf{Commit}\ T_2,\ W_1(Y),\ \textbf{Abort}\ T_1$$

Which of the following best describes S?
A. Recoverable and Cascadeless
B. Recoverable but Cascading
C. Non-Recoverable
D. Strict

Problem 278 ***[MCQ]*** *Which of the following is the **precise definition** of a **non-recoverable** schedule?*
A. A transaction T_j reads a value written by an uncommitted T_i.
B. A transaction T_j reads a value written by T_i, and T_j commits before T_i commits (or T_i subsequently aborts).
C. Two transactions write the same data item without any intervening read.
D. A transaction reads a data item that was written by a transaction that has already aborted.

Problem 279 ***[MSQ]*** *Which of the following schedules are **non-recoverable**?*
A. $R_1(X),\ W_1(X),\ R_2(X),\ C_2,\ C_1$
B. $R_1(X),\ W_1(X),\ R_2(X),\ C_2,\ A_1$ *(A_1 denotes abort of T_1)*
C. $W_1(X),\ R_2(X),\ C_1,\ C_2$
D. $W_1(X),\ R_2(X),\ C_2,\ W_1(Y),\ C_1$

Problem 280 ***[MCQ]*** *In a cascading rollback triggered by the abort of T_1, which transactions must be rolled back?*
A. Only T_1.
B. All transactions that read values written by T_1, and recursively, all transactions that depend on those.
C. All active (uncommitted) transactions in the system.
D. All transactions that wrote to any data item that T_1 also wrote to.

Problem 281 ***[MCQ]*** *Consider the schedule:*

$$S : W_1(X),\ R_2(X),\ W_2(Y),\ R_3(Y),\ W_3(Z),\ R_4(Z)$$

If T_1 is aborted (assume none have committed yet), which transactions are forced to roll back due to cascading?
A. T_2 only.
B. T_2 and T_3 only.
C. T_2, T_3, and T_4.
D. No cascading occurs because cascading rollback only applies after a commit.

Problem 282 ***[MSQ]*** *Which of the following schedules are* ***recoverable****?*
A. $W_1(X),\ R_2(X),\ C_1,\ C_2$
B. $W_1(X),\ R_2(X),\ C_2,\ C_1$
C. $W_1(X),\ C_1,\ R_2(X),\ C_2$
D. $R_1(X),\ W_2(X),\ C_2,\ W_1(X),\ C_1$

Problem 283 ***[MCQ]*** *A schedule satisfies* ***Avoiding Cascading Aborts (ACA)*** *if and only if:*
A. Every transaction reads only the initial values of all data items.
B. For every read $R_i(X)$*: if* X *was written by some* T_j*, then* T_j *has committed before* $R_i(X)$ *occurs.*
C. No two transactions access the same data item.
D. Every transaction acquires all its locks before any other transaction begins.

Problem 284 ***[MCQ]*** *Consider the schedule:*

$$S : W_1(X),\ C_1,\ R_2(X),\ W_2(Y),\ C_2,\ R_3(Y),\ C_3$$

Which recovery property does S ***satisfy****?*
A. Recoverable only (not ACA).
B. ACA (Cascadeless) but not Strict.
C. Strict.
D. Non-Recoverable.

Problem 285 ***[MSQ]*** *Which of the following schedules are* ***cascadeless (ACA)****?*
A. $W_1(X),\ C_1,\ R_2(X),\ C_2$
B. $W_1(X),\ R_2(X),\ C_1,\ C_2$
C. $W_1(X),\ W_2(X),\ C_1,\ C_2,\ R_3(X),\ C_3$
D. $R_1(X),\ W_2(X),\ C_2,\ R_1(X),\ C_1$

Problem 286 ***[MCQ]*** *A schedule is said to be* ***Strict*** *if and only if:*
A. No transaction reads a data item written by an uncommitted transaction.
B. No transaction reads or writes *a data item* X *written by transaction* T_j *until* T_j *has either committed or aborted.*
C. All transactions commit in the same order in which they acquired their first lock.
D. No transaction is allowed to write a data item that has been read by another active transaction.

Problem 287 ***[MCQ]*** *Consider the schedule:*

$$S : W_1(X),\ C_1,\ W_2(X),\ R_3(X),\ C_3,\ C_2$$

Is S ***Strict****?*
A. Yes, because $R_3(X)$ *occurs after* C_1 *and* $W_2(X)$*.*
B. No, because $R_3(X)$ *reads the value written by* T_2*, which has not yet committed at the time of* $R_3(X)$*.*
C. No, because $W_2(X)$ *overwrites* T_1*'s value before* T_1*'s effect is propagated.*
D. Yes, because all writes are separated by at least one other operation.

Problem 288 ***[MSQ]*** *Which of the following statements about* ***strict schedules*** *are correct?*
A. Every strict schedule is also cascadeless (ACA).
B. Every cascadeless (ACA) schedule is also strict.
C. Strict schedules permit simple recovery on abort.
D. The Strict 2PL protocol guarantees that all generated schedules are strict.

Problem 289 ***[MCQ]*** *Which of the following* ***correctly*** *represents the subset hierarchy of schedule classes, from* most restrictive *(smallest set) to* least restrictive *(largest set)?*
A. Serial $\subsetneq$ *Strict* $\subsetneq$ *ACA* $\subsetneq$ *Recoverable* $\subsetneq$ *All Schedules*

B. Strict $\subsetneq$ Serial $\subsetneq$ ACA $\subsetneq$ Recoverable $\subsetneq$ All Schedules
C. Serial $\subsetneq$ ACA $\subsetneq$ Strict $\subsetneq$ Recoverable $\subsetneq$ All Schedules
D. ACA $\subsetneq$ Strict $\subsetneq$ Serial $\subsetneq$ Recoverable $\subsetneq$ All Schedules

Problem 290 ***[MSQ]*** *Consider the schedule:*

$$S : R_1(X),\ W_2(X),\ C_2,\ W_1(X),\ C_1$$

Which of the following properties does S ***satisfy****?*
A. Recoverable
B. ACA (Cascadeless)
C. Strict
D. Conflict Serializable

Problem 291 ***[NAT]*** *Consider the following four properties: (1) Recoverable, (2) Cascadeless (ACA), (3) View Serializable, (4) Conflict Serializable. A system guarantees that all generated schedules are* ***Strict****. How many of the four listed properties are* automatically *also guaranteed solely by the Strict property?*

Problem 292 ***[MCQ]*** *In a two-mode locking system (Shared S and Exclusive X), data item Q is currently held by T_1 under an S-lock. Which of the following requests will be* ***denied****?*
A. An S-lock request on Q by a different transaction T_2.
B. An X-lock request on Q by T_2.
C. An upgrade of T_1's S-lock to an X-lock by T_1 itself.
D. Both B and C.

Problem 293 ***[MCQ]*** *A* ***simple lock-based protocol*** *(without the two-phase rule) can still produce:*
A. Deadlocks only.
B. Non-serializable schedules only.
C. Both deadlocks and non-serializable schedules.
D. Neither deadlocks nor non-serializable schedules, but may cause starvation.

Problem 294 ***[MSQ]*** *Which of the following statements about* ***lock-based concurrency control*** *are correct?*
A. Granting only X-locks (no S-locks) trivially ensures serializability but severely reduces concurrency.
B. A deadlock can occur even under Strict 2PL.
C. Lock upgrading (S-lock to X-lock on the same item) is permitted in the growing phase of Basic 2PL.
D. Starvation is possible in lock-based systems if the lock manager uses a non-fair (LIFO) waiting policy.

Problem 295 ***[NAT]*** *Four transactions T_1, T_2, T_3, T_4 each request a* ***Shared (S) lock*** *on data item X. The lock manager grants S-locks to compatible requesters simultaneously. How many S-locks on X can be held* ***concurrently*** *(at the same instant)?*

Problem 296 ***[MCQ]*** *Which statement* ***correctly*** *distinguishes* ***Strict 2PL*** *from* ***Rigorous 2PL****?*
A. Strict 2PL releases all locks (S and X) only at commit/abort; Rigorous 2PL releases X-locks only at commit/abort.
B. Strict 2PL releases X-locks only at commit/abort (S-locks may be released earlier); Rigorous 2PL releases all *locks only at commit/abort.*
C. Strict 2PL prevents deadlocks; Rigorous 2PL does not.
D. Rigorous 2PL does not guarantee conflict serializability; Strict 2PL does.

Problem 297 ***[MCQ]*** *Consider the following lock/unlock sequence for transaction T:*

$$\textit{lock-S}(A),\ \textit{lock-X}(B),\ \textit{unlock}(A),\ \textit{lock-S}(C),\ \textit{unlock}(B),\ \textit{unlock}(C)$$

*Does T follow the **Two-Phase Locking (2PL)** protocol?*
A. Yes, because the growing phase ends only after all locks are acquired.
B. No, because T releases the lock on A and then acquires a new lock on C, violating the two-phase rule.
C. Yes, because T acquires only S and X locks.
D. No, because T holds both an S-lock and an X-lock simultaneously.

Problem 298 ***[MSQ]*** *Which of the following are **guaranteed by Strict 2PL** but **NOT by Basic 2PL**?*
A. Conflict serializability of the generated schedules.
B. Freedom from cascading rollbacks (ACA property).
C. Strictness of the generated schedules.
D. Deadlock prevention.

Problem 299 ***[NAT]*** *Transaction T performs the following operations in order: $R(A)$, $R(B)$, $W(A)$, $R(C)$, $W(B)$, $W(C)$. Under **Basic 2PL**, T acquires an S-lock immediately before each read and an X-lock immediately before each write on first access; subsequent accesses to the same item use the already-held lock (upgrade if necessary). The lock point is the step after which no new lock or upgrade is acquired. Which operation number (counting from 1) corresponds to the **lock point**?*

Problem 300 ***[MSQ]*** *Which of the following is/are **Deadlock Free** ?*
A. Conservative 2PL.
B. Basic 2PL.
C. Strict 2PL.
D. Rigorous 2PL.

Problem 301 ***[MCQ]*** *In Multiple Granularity Locking (MGL), can an **IX** lock be granted on a node N if another transaction already holds an **S-lock** on the* same *node N?*
A. Yes, because IX and S are both compatible at coarse granularity.
*B. No, because S and IX are **incompatible** (S implies reading the whole subtree; IX implies exclusive writes in the subtree).*
C. Yes, because IX only affects descendant nodes, never node N itself.
D. Only if the granularity is at the database (root) level.

Problem 302 ***[MCQ]*** *A transaction T wants to **update a single tuple** in a relation. In MGL (hierarchy: Database $\rightarrow$ Relation $\rightarrow$ Tuple), what is the **correct** sequence of lock acquisitions from root to leaf?*
A. IX on Database $\rightarrow$ IX on Relation $\rightarrow$ X on Tuple
B. X on Database $\rightarrow$ X on Relation $\rightarrow$ X on Tuple
C. IS on Database $\rightarrow$ IS on Relation $\rightarrow$ X on Tuple
D. SIX on Database $\rightarrow$ SIX on Relation $\rightarrow$ X on Tuple

Problem 303 ***[MSQ]*** *Which of the following statements about the **SIX** (Shared-Intention-Exclusive) lock mode are **correct**?* (Select all that apply.)
A. A SIX lock on node N means: N's entire subtree is read, and some *descendants will be exclusively written.*
B. SIX is compatible with an IS-lock held by another transaction on the same node.
C. SIX is compatible with an S-lock held by another transaction on the same node.
D. SIX is compatible with an IX-lock held by another transaction on the same node.

Problem 304 ***[NAT]*** *In the MGL compatibility matrix for the five lock modes $\{IS,\ IX,\ S,\ SIX,\ X\}$, how many **ordered** pairs (L_1, L_2) are **compatible** (i.e., both grants can coexist)? Count all $5 \times 5 = 25$ ordered pairs and mark each as compatible or not using the standard MGL compatibility table. Enter the integer count.*

Problem 305 ***[MCQ]*** *Graph-based (tree) protocols provide an advantage over 2PL in which respect?*
A. They do not require transactions to know their data set in advance.
B. They guarantee deadlock freedom without needing a wait-for graph.

C. They produce recoverable schedules without additional mechanisms.
D. They do not require the database items to be partially ordered.

Problem 306 ***[MCQ]*** *In the **tree protocol**, transaction T wishes to lock data item D whose parent in the database tree is P. Which of the following is a **necessary and sufficient** condition (beyond D not being locked incompatibly by another transaction) for T to acquire a lock on D?*
A. T must currently hold a lock on every ancestor of D in the tree.
B. T must currently *hold a lock on P.*
C. T must have at some earlier point *locked P (even if it has since been released).*
D. Either B or C (holding P now, or having held it previously).

Problem 307 ***[MSQ]*** *Which of the following are known **limitations** of the tree protocol compared to 2PL?*
A. The tree protocol may force a transaction to lock data items it does not actually need (phantom locks).
B. The tree protocol does not guarantee conflict serializability.
C. The tree protocol may generate non-recoverable schedules.
D. The tree protocol requires a priori knowledge of the database's tree (partial) ordering.

Problem 308 ***[MCQ]*** *In the tree locking protocol, once transaction T **unlocks** data item Q, which rule applies?*
A. T may re-lock Q later in the same transaction, provided Q's parent is currently locked by T.
*B. T may **never** re-lock Q during the same transaction.*
C. T may re-lock Q only if it has re-locked all of Q's ancestors first.
D. T may re-lock Q after committing and restarting.

Problem 309 ***[MSQ]*** *A transaction uses the tree protocol on a tree-structured database. Which of the following statements are **true**?*
A. The resulting schedule is always conflict serializable.
B. The resulting schedule is always recoverable without additional mechanisms.
C. A data item may be unlocked by the transaction before *the transaction commits.*
D. The first lock request in the transaction can be on any *node in the tree.*

Problem 310 ***[NAT]*** *Two transactions T_1 and T_2 use the tree protocol on a tree with nodes $\{R, A, B, C\}$ where R is the root, A and B are children of R, and C is a child of B. T_1 needs to access $\{A, C\}$ and T_2 needs to access $\{B, C\}$. What is the **total minimum number of lock acquisitions** across both transactions to access all required items?*

Problem 311 ***[MCQ]*** *In the basic **Timestamp Ordering (TO)** protocol, transaction T_i issues $R_i(X)$. Let $TS(T_i)$ denote the timestamp of T_i and $W\text{-}TS(X)$ the write timestamp of X. Under what condition is $R_i(X)$ **rejected** and T_i rolled back?*
A. $TS(T_i) > W\text{-}TS(X)$
B. $TS(T_i) \geq W\text{-}TS(X)$
C. $TS(T_i) < W\text{-}TS(X)$
D. $TS(T_i) = W\text{-}TS(X)$

Problem 312 ***[MCQ]*** *Under the **Thomas Write Rule**, transaction T_i issues $W_i(X)$ with $TS(T_i) < W\text{-}TS(X)$ and $TS(T_i) \geq R\text{-}TS(X)$. The Thomas Write Rule dictates:*
A. Roll back T_i immediately.
B. Execute the write and update $W\text{-}TS(X) \leftarrow TS(T_i)$.
*C. **Ignore** (skip) the write—do* not *execute it and do* not *roll back T_i.*
D. Delay T_i until $W\text{-}TS(X)$ decreases below $TS(T_i)$.

Problem 313 ***[MSQ]*** *Which of the following statements about the **basic Timestamp Ordering (TO)** protocol are **true**?*

A. The protocol is completely deadlock-free.
B. The protocol guarantees conflict serializability.
C. The protocol may cause starvation if the same transaction is repeatedly rolled back and restarted with a new timestamp.
D. Schedules produced by basic TO are always recoverable.

Problem 314 ***[NAT]*** *Consider transactions T_1 (TS = 1), T_2 (TS = 2), T_3 (TS = 3). The schedule proceeds as follows:*

$$R_1(X),\ W_3(X),\ R_2(X),\ W_2(Y),\ R_3(Y)$$

*Initially $R\text{-}TS(X) = W\text{-}TS(X) = R\text{-}TS(Y) = W\text{-}TS(Y) = 0$. Apply the basic **Timestamp Ordering** protocol step by step. How many individual operations trigger a **transaction rollback**? Enter the integer value.*

Problem 315 ***[MCQ]*** *Consider three transactions with timestamps $TS(T_1) = 100$, $TS(T_2) = 200$, $TS(T_3) = 300$. The current timestamp values on data item X are $R\text{-}TS(X) = 150$ and $W\text{-}TS(X) = 250$. T_1 issues $R_1(X)$. T_2 issues $W_2(X)$. T_3 issues $R_3(X)$. Under the **Basic Timestamp Ordering** protocol, which operations are **rejected** (causing a rollback)?*
A. $R_1(X)$ only.
B. $W_2(X)$ only.
C. $R_1(X)$ and $W_2(X)$.
D. $R_1(X)$ and $R_3(X)$.

Problem 316 ***[MCQ]*** *Under the **Basic Timestamp Ordering** protocol, transaction T_i issues $W_i(X)$. The current timestamps on X are $R\text{-}TS(X) = r$ and $W\text{-}TS(X) = w$, and $TS(T_i) = t$. The write is **rejected and** T_i **is rolled back** when:*
A. $t < w$ only.
B. $t < r$ only.
*C. $t < r$ **OR** $t < w$ (i.e., $t < \max(r, w)$).*
*D. $t < r$ **AND** $t < w$.*

Problem 317 ***[MCQ]*** *Transactions T_1 (TS=10), T_2 (TS=20), T_3 (TS=30). Initially $R\text{-}TS(X) = 0,\ W\text{-}TS(X) = 0$. The following operations are issued in order:*

$$W_3(X), \quad W_1(X), \quad R_2(X)$$

*Under the **Thomas Write Rule**, what happens at step $W_1(X)$?*
A. $W_1(X)$ is executed and $W\text{-}TS(X)$ is updated to 10.
B. T_1 is rolled back because $TS(T_1) < W\text{-}TS(X) = 30$.
*C. $W_1(X)$ is **ignored** (skipped) but T_1 is **not** rolled back.*
*D. $W_1(X)$ is ignored **and** T_1 is rolled back.*

Problem 318 ***[MCQ]*** *A schedule is produced by the Thomas Write Rule but is **not** producible by the Basic Timestamp Ordering protocol. Which of the following **best characterises** such a schedule?*
A. It is conflict serializable in the timestamp order, with some reads reordered.
*B. It contains **blind writes** that are silently skipped, making the schedule view serializable in the timestamp order while the precedence graph may have a cycle.*
C. It is non-recoverable because skipped writes leave dirty values on disk.
D. It is always strict because no dirty data is ever read.

Problem 319 ***[MCQ]*** *In the **Wait-Die** scheme, transaction T_i requests a lock held by T_j. Under what condition does T_i **wait**?*
A. $TS(T_i) > TS(T_j)$ (i.e., T_i is younger *than T_j).*
B. $TS(T_i) < TS(T_j)$ (i.e., T_i is older *than T_j).*
C. $TS(T_i) = TS(T_j)$.
D. T_i always waits regardless of timestamps.

Problem 320 ***[MCQ]*** *In the* ***Wound-Wait*** *scheme, T_i requests a lock held by T_j. If $TS(T_i) < TS(T_j)$ (i.e., T_i is* older*), what action is taken?*
A. T_i waits for T_j to release the lock.
B. T_i is rolled back (dies).
C. T_j is preemptively rolled back (wounded) and T_i proceeds.
D. Both T_i and T_j are rolled back.

Problem 321 ***[MCQ]*** *Transactions with timestamps $TS(T_1) = 5,\ TS(T_2) = 10,\ TS(T_3) = 15$ (smaller = older). T_3 holds a lock on X. First, T_1 requests X; then T_2 requests X. Under* ***Wait-Die****, what happens?*
A. T_1 waits; T_2 dies.
B. T_1 dies; T_2 waits.
C. Both T_1 and T_2 wait.
D. Both T_1 and T_2 die.

Problem 322 ***[MSQ]*** *Which of the following statements about* ***Wait-Die*** *and* ***Wound-Wait*** *are* ***correct****?*
A. Both schemes are ***non-preemptive*** *in the sense that running transactions are never forcibly rolled back.*
B. In Wait-Die, only the younger *(higher TS) transaction can be rolled back.*
C. In Wound-Wait, a younger *requesting transaction* ***always waits*** *if the holder is older.*
D. Both schemes guarantee ***freedom from deadlock*** *because they impose a total order on lock requests.*

Problem 323 ***[NAT]*** *Transactions T_1 (TS=10), T_2 (TS=20), T_3 (TS=30), T_4 (TS=40) (smaller TS = older). Lock conflicts occur in the following order:*

1. *T_4 requests lock held by T_1*
2. *T_3 requests lock held by T_2*
3. *T_2 requests lock held by T_1*
4. *T_4 requests lock held by T_3*

Apply the ***Wait-Die*** *scheme to each conflict independently. How many transactions are* ***rolled back (die)*** *in total?*

Problem 324 ***[MCQ]*** *The* ***Write-Ahead Logging (WAL)*** *rule states that:*
A. A transaction must write its log records to disk after *the actual data pages are flushed.*
B. Before any data item X is written to disk (stable storage), all log records pertaining to X must first be flushed to the log on stable storage.
C. A transaction may commit only after all its data pages have been flushed to disk.
D. The log must be written to disk at the same time (atomically) as the data pages.

Problem 325 ***[MSQ]*** *Consider the following log (written in order, top to bottom):*

$$\langle T_1, \textit{begin} \rangle$$

$$\langle T_1, A, 100, 150 \rangle$$

$$\langle T_2, \textit{begin} \rangle$$

$$\langle T_2, B, 200, 250 \rangle$$

$$\langle T_1, \textit{commit} \rangle$$

$$\langle T_3, \textit{begin} \rangle$$

$$\langle T_3, C, 300, 350 \rangle$$

— CRASH —

*The system uses **UNDO-REDO** logging. Which of the following actions must be taken during recovery?*
*A. **REDO** T_1: write $A = 150$.*
*B. **UNDO** T_2: write $B = 200$.*
*C. **UNDO** T_3: write $C = 300$.*
*D. **REDO** T_2: write $B = 250$.*

Problem 326 ***[MCQ]*** *A **checkpoint** record is added to the log. During recovery using UNDO-REDO, which transactions **need not be considered** for redo or undo?*
A. All transactions that started after the most recent checkpoint.
*B. All transactions that **committed before** the most recent checkpoint* and *whose dirty pages were all flushed to disk as part of the checkpoint.*
C. All transactions that were active at the time of the checkpoint.
D. All transactions that appeared in any prior checkpoint.

Problem 327 ***[NAT]*** *Consider the following log for UNDO-only recovery (steal/force policy):*

LSN	***Log Record***	
1	$\langle T_1, \textit{begin} \rangle$	
2	$\langle T_1, A, 50 \rangle$	*(A's old value is 50; new value written to disk)*
3	$\langle T_1, B, 80 \rangle$	*(B's old value is 80)*
4	$\langle T_2, \textit{begin} \rangle$	
5	$\langle T_2, C, 120 \rangle$	*(C's old value is 120)*
6	$\langle T_1, \textit{commit} \rangle$	
7	$\langle T_2, D, 200 \rangle$	*(D's old value is 200)*
	— CRASH —	

*During recovery, how many **UNDO** operations are performed?*

8.6 GATE PYQs

GATEPYQ 146 *Consider concurrent execution of two transactions T_1 and T_2 in a DBMS, both of which access a data object A. For these two transactions to not conflict on A, which one of the following statements must be true?*

A. Both T_1 and T_2 only read A
B. T_1 reads A and T_2 writes A
C. T_1 writes A and T_2 reads A
D. Both T_1 and T_2 write A
GATE CSE 2026

GATEPYQ 147 *Consider the database transactions T_1 and T_2, and data items X and Y.*

***Transaction** T_1: $R_1(X)$, $W_1(Y)$, $R_1(X)$, $W_1(X)$, COMMIT(T_1)*
***Transaction** T_2: $W_2(X)$, $W_2(Y)$, COMMIT(T_2)*
Which of the following schedule(s) is/are conflict serializable?
A. $R_1(X), W_2(X), W_1(Y), W_2(Y), R_1(X), W_1(X)$, COMMIT$(T_2)$, COMMIT$(T_1)$
B. $W_2(X), R_1(X), W_2(Y), W_1(Y), R_1(X)$, COMMIT$(T_2)$, $W_1(X)$, COMMIT(T_1)
C. $R_1(X), W_1(Y), W_2(X), W_2(Y), R_1(X), W_1(X)$, COMMIT$(T_1)$, COMMIT$(T_2)$
D. $W_2(X), R_1(X), W_1(Y), W_2(Y), R_1(X)$, COMMIT$(T_2)$, $W_1(X)$, COMMIT(T_1)
GATE CSE 2025

GATEPYQ 148 *An audit of a banking transactions system has found that on an earlier occasion, two joint holders of account A attempted simultaneous transfers of Rs. 10000 each from account A to account B. Both transactions read the same value, Rs. 11000, as the initial balance in A and were allowed to go through. B was credited Rs. 10000 twice. A was debited only once and ended up with a balance of Rs. 1000.*

Which of the following properties is/are certain to have been violated by the system?
A. Atomicity
B. Consistency
C. Isolation
D. Durability
GATE CSE 2025

GATEPYQ 149 *A schedule of three database transactions T_1, T_2, and T_3 is shown. $R_i(A)$ and $W_i(A)$ denote read and write of data item A by transaction T_i, $i = 1, 2, 3$. The transaction T_1 aborts at the end. Which other transaction(s) will be required to be rolled back?*

$R_1(X)$ $W_1(Y)$ $R_2(X)$ $R_2(Y)$ $R_3(Y)$ ABORT(T_1)
A. Only T_2
B. Only T_3
C. Both T_2 and T_3
D. Neither T_2 nor T_3
GATE CSE 2025

GATEPYQ 150 *Which of the following statements about the Two Phase Locking (2PL) protocol is/are TRUE?*

A. 2PL permits only serializable schedules

B. With 2PL, a transaction always locks the data item being read or written just before every operation and always releases the lock just after the operation

C. With 2PL, once a lock is released on any data item inside a transaction, no more locks on any data item can be obtained inside that transaction

D. A deadlock is possible with 2PL
GATE CSE 2024

GATEPYQ 151 *Once the DBMS informs the user that a transaction has been successfully completed, its effect should persist even if the system crashes before all its changes are reflected on disk. This property is called*

A. durability
B. atomicity
C. consistency
D. isolation
GATE CSE 2024

GATEPYQ 152 *Consider the following read-write schedule S over three transactions T_1, T_2, and T_3, where the subscripts in the schedule indicate transaction IDs:*

$$S: \ r_1(z);\ w_1(z);\ r_2(x);\ r_3(y);\ w_3(y);\ r_2(y);\ w_2(x);\ w_2(y);$$

Which of the following transaction schedules is/are conflict equivalent to S?
A. $T_1\ T_2\ T_3$
B. $T_1\ T_3\ T_2$
C. $T_3\ T_2\ T_1$
D. $T_3\ T_1\ T_2$
GATE CSE 2024

GATEPYQ 153 *Let $R_i(z)$ and $W_i(z)$ denote read and write operations on a data element z by a transaction T_i, respectively. Consider the schedule S with four transactions.*

$$S: \ R_4(x)\ R_2(x)\ R_3(x)\ R_1(y)\ W_1(y)\ W_2(x)\ W_3(y)\ R_4(y)$$

Which one of the following serial schedules is conflict equivalent to S?
A. $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$
B. $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$
C. $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$
D. $T_3 \rightarrow T_1 \rightarrow T_4 \rightarrow T_2$
GATE CSE 2022

GATEPYQ 154 *Let S be the following schedule of operations of three transactions T_1, T_2 and T_3 in a relational database system:*

$$R_2(Y),\ R_1(X),\ R_3(Z),\ R_1(Y),\ W_1(X),\ R_2(Z),\ W_2(Y),\ R_3(X),\ W_3(Z)$$

Consider the statements P and Q below:
P: S is conflict-serializable.
Q: If T_3 commits before T_1 finishes, then S is recoverable.
Which one of the following choices is correct?
A. Both P and Q are true
B. P is true and Q is false
C. P is false and Q is true
D. Both P and Q are false
GATE CSE 2021

GATEPYQ 155 *Let $r_i(z)$ and $w_i(z)$ denote read and write operations respectively on a data item z by a transaction T_i. Consider the following two schedules.*

$$S_1 : r_1(x),\ r_1(y),\ r_2(x),\ r_2(y),\ w_2(y),\ w_1(x)$$

$$S_2 : r_1(x),\ r_2(x),\ r_2(y),\ w_2(y),\ r_1(y),\ w_1(x)$$

Which one of the following options is correct?
A. S_1 is conflict serializable, and S_2 is not conflict serializable
B. S_1 is not conflict serializable, and S_2 is conflict serializable
C. Both S_1 and S_2 are conflict serializable
D. Neither S_1 nor S_2 is conflict serializable
GATE CSE 2021

GATEPYQ 156 *Suppose a database system crashes again while recovering from a previous crash. Assume checkpointing is not done by the database either during the transactions or during recovery.*
Which of the following statements is/are correct?
A. The same undo and redo list will be used while recovering again
B. The system cannot recover any further
C. All the transactions that are already undone and redone will not be recovered again
D. The database will become inconsistent
GATE CSE 2021

GATEPYQ 157 *Consider a schedule of transactions T_1 and T_2:*

T_1	RA			RC		WD		WB	Commit	
T_2		RB	WB		RD		WC			Commit

Here, RX stands for "Read(X)" and WX stands for "Write(X)". Which one of the following schedules is conflict equivalent to the above schedule?

A.

T_1				RA	RC	WD	WB		Commit	
T_2	RB	WB	RD					WC		Commit

B.

T_1	RA	RC	WD	WB					Commit	
T_2					RB	WB	RD	WC		Commit

C.

T_1	RA	RC	WD				WB		Commit	
T_2				RB	WB	RD		WC		Commit

D.

T_1					RA	RC	WD	WB	Commit	
T_2	RB	WB	RD	WC						Commit

GATEPYQ 158 *Consider the following two statements about database transaction schedules:*

I. Strict two-phase locking protocol generates conflict serializable schedules that are also recoverable.

II. Timestamp-ordering concurrency control protocol with Thomas' Write Rule can generate view serializable schedules that are not conflict serializable.

Which of the above statements is/are TRUE?

A. I only

B. II only

C. Both I and II

D. Neither I nor II

GATE CSE 2019

GATEPYQ 159 *Two transactions T_1 and T_2 are given as:*

T_1: $r_1(X)$ $w_1(X)$ $r_1(Y)$ $w_1(Y)$

T_2: $r_2(Y)$ $w_2(Y)$ $r_2(Z)$ $w_2(Z)$

where $r_i(V)$ denotes a read operation by transaction T_i on a variable V and $w_i(V)$ denotes a write operation by transaction T_i on a variable V. The total number of conflict serializable schedules that can be formed by T_1 and T_2 is

GATE CSE 2017

GATEPYQ 160 *In a database system, unique timestamps are assigned to each transaction using Lamport's logical clock. Let TS(T_1) and TS(T_2) be the timestamps of transactions T_1 and T_2, respectively. Suppose T_1 holds a lock on the resource R, and T_2 has requested a conflicting lock on the same resource R. The following algorithm is used to prevent deadlocks in the database system assuming that a killed transaction is restarted with the same timestamp. Assume any transaction that is not killed terminates eventually.*

if $TS(T_2) < TS(T_1)$ *then*

T_1 *is killed*

else T_2 waits.

Which of the following is TRUE about the database system that uses the above algorithm to prevent deadlocks?

A. The database system is both deadlock-free and starvation-free.
B. The database system is deadlock-free, but not starvation-free.
C. The database system is starvation-free but not deadlock-free.
D. The database system is neither deadlock-free nor starvation-free.
GATE CSE 2017

GATEPYQ 161 *Consider the following database schedule with two transactions, T_1 and T_2.*

$S = r_2(X); r_1(X); r_2(Y); w_1(X); r_1(Y); w_2(X); a_1; a_2$

where $r_i(Z)$ denotes a read operation by transaction T_i on a variable Z, $w_i(Z)$ denotes a write operation by T_i on a variable Z, and a_i denotes an abort by transaction T_i.

Which one of the following statements about the above schedule is TRUE?

A. S is non-recoverable
B. S is recoverable, but has a cascading abort
C. S does not have a cascading abort
D. S is strict
GATE CSE 2016

GATEPYQ 162 *Suppose a database schedule S involves transactions $T_1, \ldots, T_n$. Construct the precedence graph of S with vertices representing the transactions and edges representing the conflicts. If S is serializable, which one of the following orderings of the vertices of the precedence graph is guaranteed to yield a serial schedule?*

A. Topological order
B. Depth-first order
C. Breadth-first order
D. Ascending order of transaction indices
GATE CSE 2016

GATEPYQ 163 *Consider the following two-phase locking protocol. Suppose a transaction T accesses (for read or write operations) a certain set of objects $\{O_1, \ldots, O_k\}$. This is done in the following manner:*

Step 1. T acquires exclusive locks to $O_1, \ldots, O_k$ in increasing order of their addresses. Step 2. The required operations are performed. Step 3. All locks are released.

This protocol will:

A. Guarantee serializability and deadlock-freedom
B. Guarantee neither serializability nor deadlock-freedom
C. Guarantee serializability but not deadlock-freedom
D. Guarantee deadlock-freedom but not serializability
GATE CSE 2016

GATEPYQ 164 *Which one of the following is NOT a part of the ACID properties of database transactions?*

A. Atomicity
B. Consistency
C. Isolation
D. Deadlock-freedom
GATE CSE 2016

GATEPYQ 165 *Consider the following partial Schedule S involving two transactions $T1$ and $T2$. Only the read and the write operations have been shown. The read operation on data item P is denoted by $read(P)$ and the write operation on data item P is denoted by $write(P)$.*

Time instance	Transaction-id	
	$T1$	$T2$
1	$read(A)$	
2	$write(A)$	
3		$read(C)$
4		$write(C)$
5		$read(B)$
6		$write(B)$
7		$read(A)$
8		$commit$
9	$read(B)$	

Suppose that the transaction $T1$ fails immediately after time instance 9. Which one of the following statements is correct?

A. $T2$ must be aborted and then both $T1$ and $T2$ must be re-started to ensure transaction atomicity
B. Schedule S is non-recoverable and cannot ensure transaction atomicity
C. Only $T2$ must be aborted and then re-started to ensure transaction atomicity
D. Schedule S is recoverable and can ensure atomicity and nothing else needs to be done
GATE CSE 2015

GATEPYQ 166 *Consider a simple checkpointing protocol and the following set of operations in the log:*

(start, T4); (write, T4, y, 2, 3); (start, T1); (commit, T4); (write, T1, z, 5, 7); (checkpoint); (start, T2); (write, T2, x, 1, 9); (commit, T2); (start, T3); (write, T3, z, 7, 2)

If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list?

A. Undo: T3, T1; Redo: T2
B. Undo: T3, T1; Redo: T2, T4
C. Undo: none; Redo: T2, T4, T3, T1
D. Undo: T3, T1, T4; Redo: T2
GATE CSE 2015

GATEPYQ 167 *Consider the following transaction involving two bank accounts x and y:*

$$read(x); \quad x := x - 50; \quad write(x); \quad read(y); \quad y := y + 50; \quad write(y)$$

The constraint that the sum of the accounts x and y should remain constant is that of

A. Atomicity
B. Consistency
C. Isolation
D. Durability
GATE CSE 2015

GATEPYQ 168 *Consider the transactions $T1$, $T2$, and $T3$ and the schedules $S1$ and $S2$ given below:*

$T1 : r1(x);\ r1(Z);\ w1(x);\ w1(Z)$
$T2 : r2(x);\ r2(Z);\ w2(Z)$
$T3 : r3(x);\ r3(x);\ w3(Y)$
$S1 : r1(x);\ r3(Y);\ r3(x);\ r2(Y);\ r2(Z);\ w3(Y);\ w2(Z);\ r1(Z);\ w1(x);\ w1(Z)$

$S2 : r1(x);\ r3(Y);\ r2(Y);\ r3(x);\ r1(Z);\ r2(Z);\ w3(Y);\ w1(x);\ w2(Z);\ w1(Z)$

Which one of the following statements about the schedules is TRUE?

A. Only $S1$ is conflict-serializable

B. Only $S2$ is conflict-serializable

C. Both $S1$ and $S2$ are conflict-serializable

D. Neither $S1$ nor $S2$ is conflict-serializable

GATE CSE 2014

GATEPYQ 169 *Consider the following schedule S of transactions $T1$, $T2$, $T3$, $T4$:*

T1	*T2*	*T3*	*T4*
	Reads(X)		
		Writes(X)	
		Commit	
			Writes(X)
Commit			
	Writes(Y)		
	Reads(Z)		
	Commit		
			Reads(X)
			Reads(Y)
			Commit

Which one of the following statements is CORRECT?

A. S is conflict-serializable but not recoverable

B. S is not conflict-serializable but is recoverable

C. S is both conflict-serializable and recoverable

D. S is neither conflict-serializable nor is it recoverable

GATE CSE 2014

GATEPYQ 170 *Consider the following four schedules due to three transactions (indicated by the subscript) using read and write on a data item x, denoted by $r(x)$ and $w(x)$ respectively.*

Which one of them is conflict-serializable?

A. $r1(x);\ r2(x);\ w1(x);\ r3(x);\ w2(x)$

B. $r2(x);\ r1(x);\ w2(x);\ r3(x);\ w1(x)$

C. $r3(x);\ r2(x);\ r1(x);\ w2(x);\ w1(x)$

D. $r2(x);\ w2(x);\ r3(x);\ r1(x);\ w1(x)$

GATE CSE 2014

GATEPYQ 171 *Consider the following transactions with data items P and Q initialized to zero:*

$T1$:

$$read(P);\quad read(Q);\quad \text{if } P = 0 \text{ then } Q := Q + 1;\quad write(Q)$$

$T2$:

$$read(Q);\quad read(P);\quad \text{if } Q = 0 \text{ then } P := P + 1;\quad write(P)$$

Any non-serial interleaving of $T1$ and $T2$ for concurrent execution leads to:

A. a serializable schedule

B. a schedule that is not conflict serializable
C. a conflict serializable schedule
D. a precedence graph cannot be drawn
GATE CSE 2012

GATEPYQ 172 *Consider the following schedule for transactions $T1$, $T2$, and $T3$:*

T1	T2	T3
Read(X)		
	Read(Y)	
		Read(Y)
	Write(Y)	
Write(X)		
		Write(X)
	Read(X)	
	Write(X)	

Which one of the schedules below is the correct serialization of the above?
A. $T1 \rightarrow T3 \rightarrow T2$
B. $T2 \rightarrow T1 \rightarrow T3$
C. $T2 \rightarrow T3 \rightarrow T1$
D. $T3 \rightarrow T1 \rightarrow T2$
GATE CSE 2010

GATEPYQ 173 *Which of the following concurrency control protocols ensure both conflict serializability and freedom from deadlock?*
I. 2-phase locking
II. Time-stamp ordering
A. I only
B. II only
C. Both I and II
D. Neither I nor II
GATE CSE 2010

GATEPYQ 174 *Consider two transactions T_1 and T_2, and four schedules S_1, S_2, S_3, S_4 of T_1 and T_2 as given below:*
T_1: $R_1[x]\, W_1[x]\, W_1[y]$
T_2: $R_2[x]\, R_2[x]\, W_1[y]$
S_1: $R_1[x]\, R_2[x]\, R_2[y]\, W_1[x]\, W_1[y]\, W_2[y]$
S_2: $R_1[x]\, R_2[x]\, R_2[y]\, W_1[x]\, W_2[y]\, W_1[y]$
S_3: $R_1[x]\, W_1[x]\, R_2[x]\, W_1[y]\, R_2[y]\, W_2[y]$
S_4: $R_2[x]\, R_2[y]\, R_1[x]\, W_1[x]\, W_1[y]\, W_2[y]$
Which of the above schedules are conflict-serializable?
A. S_1 and S_2
B. S_2 and S_3
C. S_3 only
D. S_4 only
GATE CSE 2009

GATEPYQ 175 *Consider the following schedules involving two transactions:*
T_1: $r_1(X); r_1(Y); r_2(X); r_2(Y); w_2(Y); w_1(X)$
T_2: $r_1(X); r_2(X); r_2(Y); w_2(Y); r_1(Y); w_1(X)$
Which one of the following statements is TRUE?
A. Both S_1 and S_2 are conflict serializable.
B. S_1 is conflict serializable and S_2 is not conflict serializable.
C. S_1 is not conflict serializable and S_2 is conflict serializable.
D. Both S_1 and S_2 are not conflict serializable.
GATE CSE 2007

GATEPYQ 176 *Consider the following log sequence of two transactions on a bank account with initial balance 12000, where a transfer of 2000 is made to a mortgage payment, and then a 5% interest is applied:*

1. *T1 start*
2. *T1 B old = 12000 new = 10000*
3. *T1 M old = 0 new = 2000*
4. *T1 commit*
5. *T2 start*
6. *T2 B old = 10000 new = 10500*
7. *T2 commit*

Suppose the database system crashed just before log record 7 is written. When the system is restarted, which one statement is true of the recovery procedure?
A. We must redo log record 6 to set B to 10500
B. We must undo log record 6 to set B to 10000 and then redo log records 2 and 3
C. We need not redo log records 2 and 3 because transaction T1 has committed
D. We can apply redo and undo operations in arbitrary order because they are idempotent
GATE CSE 2006

GATEPYQ 177 *Which of the following scenarios may lead to an irrecoverable error in a database system?*
A. A transaction writes a data item after it is read by an uncommitted transaction

B. A transaction reads a data item after it is read by an uncommitted transaction

C. A transaction reads a data item after it is written by a committed transaction

D. A transaction reads a data item after it is written by an uncommitted transaction

GATE CSE 2003

GATEPYQ 178 *Consider three data items $D1, D2$ and $D3$ and the following execution schedule of transactions $T1, T2$ and $T3$. In the diagram, $R(D)$ and $W(D)$ denote the actions reading and writing the data item D respectively.*

T1	T2	T3
	R(D3); R(D2); W(D2);	
		R(D2); R(D3);
R(D1); W(D1);		
		W(D2); W(D3);
	R(D1);	
R(D2); W(D2);		
	W(D1);	

Which of the following statements is correct?

A. The schedule is serializable as $T2; T3; T1$

B. The schedule is serializable as $T2; T1; T3$

C. The schedule is serializable as $T3; T2; T1$

D. The schedule is not serializable

GATE CSE 2003

8.7 Try it Yourself

Exercise 226 ***[MCQ]*** *Which of the following sequences of transaction states is **invalid** according to the standard five-state transaction state diagram?*
A. Active → Partially Committed → Committed
B. Active → Failed → Aborted
C. Active → Partially Committed → Failed → Aborted
D. Active → Committed → Aborted

Exercise 227 ***[NAT]*** *In the standard five-state transaction state diagram (Active, Partially Committed, Committed, Failed, Aborted), consider* only *the transitions that are formally defined (excluding any optional "restart" arc from Aborted back to Active). How many distinct directed transitions (edges) exist in this diagram?*

Exercise 228 ***[NAT]*** *Consider the schedule:*

$$S : R_1(A),\ W_1(A),\ R_2(A),\ R_2(B),\ W_2(B),\ C_2,\ R_3(B),\ W_1(B),\ C_1,\ C_3$$

*How many read operations in S read a value written by a transaction that is **not yet committed** at the time of that read?*

Exercise 229 ***[NAT]*** *In a system, T_1 aborts. T_2 had read data written by T_1; T_3 had read data written by T_2; T_4 had read data written by T_3; T_5 had read data written by T_4. None of T_2–T_5 have committed. What is the total number of transactions (including T_1) that must be rolled back?*

Exercise 230 ***[NAT]*** *Among the following four schedules, how many are **cascadeless (ACA)**?*

1. $W_1(X),\ R_2(X),\ C_1,\ C_2$
2. $W_1(X),\ C_1,\ R_2(X),\ C_2$
3. $W_1(X),\ R_2(X),\ C_2,\ C_1$
4. $W_1(X),\ C_1,\ W_2(X),\ R_3(X),\ C_2,\ C_3$

Enter the integer count.

Exercise 231 ***[NAT]*** *Among the following four schedules, how many are **strict**?*

1. $W_1(X),\ C_1,\ W_2(X),\ C_2$
2. $W_1(X),\ W_2(X),\ C_1,\ C_2$
3. $W_1(X),\ C_1,\ R_2(X),\ C_2$
4. $R_1(X),\ W_2(X),\ C_2,\ R_1(X),\ C_1$

Enter the integer count.

Exercise 232 ***[MCQ]*** *Conflict serializability alone **does not** guarantee which of the following properties?*
A. The schedule is equivalent (in outcome) to some serial schedule.
B. The schedule is recoverable.
C. The precedence graph of the schedule is acyclic.
D. The schedule preserves database consistency if each transaction individually does.

Exercise 233 ***[MSQ]*** *Which of the following mechanisms **directly** enforce the **Durability** property of ACID?*
A. Write-Ahead Logging (WAL / REDO logging)
B. Two-Phase Locking (2PL)
C. Shadow Paging
D. Periodic checkpointing combined with log-based recovery

Exercise 234 ***[NAT]*** *Four transactions T_1, T_2, T_3, T_4 are to be executed. How many distinct **serial** schedules exist over these four transactions?*

Exercise 235 ***[MCQ]*** *The precedence graph for schedule S over $\{T_1, T_2, T_3, T_4\}$ has edges: $T_1 \to T_2,\ T_2 \to T_3,\ T_3 \to T_4,\ T_4 \to T_2$. Which of the following is **correct**?*
A. S is conflict serializable with equivalent serial order $T_1 \to T_4 \to T_2 \to T_3$.
*B. S is **not** conflict serializable because the graph contains a cycle.*
C. S is conflict serializable with equivalent serial order $T_1 \to T_2 \to T_3 \to T_4$.
D. S is conflict serializable because T_1 has no incoming edges.

Exercise 236 ***[MSQ]*** *Consider the following schedule:*

$$S : R_1(A),\ R_2(B),\ W_1(B),\ R_3(A),\ W_2(A),\ W_3(B)$$

*Which of the following describes a **cycle** in the precedence (serializability) graph of S?*
A. $T_1 \to T_2 \to T_1$
B. $T_1 \to T_2 \to T_3 \to T_1$
C. $T_2 \to T_3 \to T_2$
D. No cycle exists; S is conflict serializable.

Exercise 237 ***[MSQ]*** *For two schedules S and S' (over the same set of transactions and operations) to be **view equivalent**, which of the following conditions must hold for every data item X?*
A. If T_i reads the initial *value of X in S, then T_i also reads the initial value of X in S'.*
B. If T_i reads the value of X written by T_j in S, then T_i reads the value of X written by T_j in S'.
C. The transaction that performs the final write *on X in S also performs the final write on X in S'.*
D. The total number of read operations on X must be the same in S and S'.

Exercise 238 ***[MCQ]*** *Consider the schedule:*

$$S :\ W_1(X),\ W_2(X),\ W_1(X),\ C_1,\ C_2$$

*This schedule is **view serializable** but **not conflict serializable**. The **key reason** it escapes conflict serializability but is still view serializable is:*
A. It has no read operations, so view equivalence conditions on reads are vacuously satisfied.
B. The writes by T_1 and T_2 commute because they write the same value.
C. The second $W_1(X)$ is a blind write that makes the precedence graph acyclic.
D. T_2 commits before T_1, automatically satisfying view equivalence.

Exercise 239 ***[MSQ]*** *For a schedule S to be **view serializable but NOT conflict serializable**, which conditions are **necessary**?*
A. The precedence graph of S must contain at least one cycle.
*B. S must contain at least one **blind write** (a write not preceded by a read of the same item in the same transaction).*
C. Every cycle in the precedence graph must involve only transactions that perform blind writes.
D. S must not contain any dirty reads.

Exercise 240 ***[MCQ]*** *Three transactions T_1, T_2, T_3 each perform only writes (no reads):*

$$S :\ W_1(X),\ W_2(Y),\ W_3(X),\ W_1(Y),\ W_2(X),\ W_3(Y)$$

*After S, which transaction's write is the **final write** on X and on Y respectively?*
A. Final write on X: T_2; Final write on Y: T_3
B. Final write on X: T_3; Final write on Y: T_1
C. Final write on X: T_2; Final write on Y: T_1
D. Final write on X: T_3; Final write on Y: T_2

Exercise 241 ***[MSQ]*** *Which of the following are **true** about view serializability?*
A. Deciding whether a schedule is view serializable is NP-complete.
B. Every conflict serializable schedule is also view serializable.
C. A schedule with no read operations is always view serializable.
*D. View serializability is a **stricter** (smaller) class than conflict serializability.*

Exercise 242 ***[MCQ]*** *Which of the following schedules is **view serializable** but **NOT conflict serializable**?*
A. $R_1(X),\ W_2(X),\ R_2(Y),\ W_1(Y),\ C_1,\ C_2$
B. $W_1(X),\ W_2(X),\ W_2(Y),\ W_1(Y),\ W_3(X)$
C. $R_1(X),\ W_1(X),\ R_2(X),\ W_2(X),\ C_1,\ C_2$
D. $W_1(X),\ R_2(X),\ W_2(X),\ C_1,\ C_2$

Exercise 243 ***[MCQ]*** *Schedule:*

$$S:\ R_1(X),\ W_2(X),\ W_1(X),\ C_1,\ C_2$$

*T_1 reads the **initial** value of X and then overwrites it. Which serial schedule is **view equivalent** to S?*
A. $T_1 \to T_2$ only.
B. $T_2 \to T_1$ only.
C. Both $T_1 \to T_2$ and $T_2 \to T_1$.
D. Neither; S is not view serializable.

Exercise 244 *Consider the schedule $S: R_1(X), W_2(X), W_2(Y), R_1(Y), C_2, C_1$.*
Which of the following is true regarding its properties?
A. S is conflict serializable and recoverable.
B. S is not conflict serializable but is recoverable.
C. S is conflict serializable but not recoverable.
D. S is neither conflict serializable nor recoverable.

Exercise 245 *In a system using the **Timestamp Ordering Protocol**, let $TS(T_1) = 50$ and $TS(T_2) = 60$. The following sequence occurs: $R_1(X), W_2(X), R_1(X)$.*
What is the result of the final $R_1(X)$ operation?
A. It is executed normally.
B. T_1 is rolled back because $TS(T_1) < W\_TS(X)$.
C. It is ignored using the Thomas Write Rule.
D. T_2 is rolled back to allow T_1 to read.

Exercise 246 *Which of the following scenarios describes a **Write-Skew** anomaly, often found in Snapshot Isolation?*
A. T_1 reads X, T_2 reads X, T_1 writes X, T_2 writes X.
B. T_1 reads X and Y, T_2 reads X and Y, T_1 writes X, T_2 writes Y.
C. T_1 writes X, T_2 reads X before T_1 commits.
D. T_1 reads X, T_2 deletes X, T_1 reads X again.

Exercise 247 *Consider the schedule $S: W_1(X), R_2(X), W_3(X), C_1, C_3, C_2$.*
Which of the following serial schedules is view equivalent to S?
A. $T_1 \to T_2 \to T_3$
B. $T_1 \to T_3 \to T_2$

C. $T_3 \to T_2 \to T_1$
D. No serial schedule is view equivalent.

Exercise 248 *In **Two-Phase Locking (2PL)**, the "Lock Point" is defined as:*
A. The moment the transaction begins execution.
B. The moment the transaction acquires its final lock.
C. The moment the transaction releases its first lock.
D. The moment the transaction commits.

Exercise 249 *A schedule* S *involves transactions* T_1, T_2, T_3*. The precedence graph has edges* $T_1 \to T_2$ *and* $T_2 \to T_3$*. If we add the operation* $W_3(X)$ *followed by* $R_1(X)$*, the resulting schedule:*
A. Remains conflict serializable.
B. Develops a cycle $T_1 \to T_2 \to T_3 \to T_1$*.*
C. Becomes view serializable only.
D. Violates the ACID property of Atomicity.

Exercise 250 ***[NAT]*** *A database tree has the following structure: Root* R *has children* A *and* B*;* A *has children* C *and* D*;* B *has children* E *and* F*;* D *has one child* G*. Transaction* T *needs to access* only *item* G*. Under the **tree protocol**, what is the **minimum** number of lock acquisitions* T *must perform to lock* G*?*

Exercise 251 ***[MCQ]*** *The tree protocol guarantees **deadlock freedom**. What is the **correct explanation**?*
A. It forces transactions to acquire all locks atomically at the start.
B. The tree imposes a total order on lock acquisitions; since all transactions acquire locks along downward paths, circular wait is impossible.
C. It uses lock timeouts to detect deadlocks and rolls back the victim.
D. It allows lock preemption: a higher-priority transaction can steal a lock from a lower-priority one.

Exercise 252 *Consider the following log at the time of a system crash:*
$\langle T_1, \textit{start}\rangle, \langle T_1, X, 100, 200\rangle, \langle T_2, \textit{start}\rangle, \langle T_2, Y, 50, 100\rangle, \langle \textit{checkpoint}\{T_1, T_2\}\rangle, \langle T_3, \textit{start}\rangle, \langle T_3, Z, 200, 300\rangle, \langle T_2, \textit{commit}\rangle$*.*
*Which transactions must be **Redone** during recovery?*
A. T_1, T_2, T_3
B. T_2 *only.*
C. T_1, T_2 *only.*
D. T_3 *only.*

Exercise 253 *In the **Tree Protocol** for graph-based locking, which of the following is TRUE?*
A. It guarantees conflict serializability and freedom from deadlock.
B. It requires all locks to be held until commit.
C. A transaction can unlock a node and then lock its child.
D. It is a variation of Strict 2PL.

Exercise 254 *Given* $S : R_1(X), R_2(Y), W_1(Y), W_2(X)$*.*
This schedule is an example of:
A. A conflict serializable schedule.
B. A deadlock-prone schedule under 2PL.
C. A recoverable schedule.
D. A view serializable schedule.

Exercise 255 *A transaction* T_1 *reads a data item* X *that was updated by* T_2*. Before* T_1 *commits,* T_2 *aborts.* T_1 *then has to be aborted. This phenomenon is known as:*
A. Lost Update.

B. Dirty Read.
C. Cascading Rollback.
D. Phantom Read.

Exercise 256 *In **Rigorous 2PL**, the throughput of the system is generally:*
A. Higher than Basic 2PL due to less waiting.
B. Lower than Basic 2PL because locks are held longer.
C. Higher than Basic 2PL because it prevents deadlocks.
D. The same as Basic 2PL.

Exercise 257 *Consider the schedule* $S : W_1(X), W_2(Y), W_1(Y), C_1, C_2$.
The equivalent serial order is:
A. $T_1 \rightarrow T_2$
B. $T_2 \rightarrow T_1$
C. Both A and B.
D. No serial order exists.

Exercise 258 *Which of the following properties of a schedule is **not** concerned with the order of read and write operations, but rather with the failure of transactions?*
A. Serializability.
B. Recoverability.
C. Atomicity.
D. Precedence.

Exercise 259 *In a database using **Immediate Update** recovery, the log record* $\langle T_i, X, V_1, V_2 \rangle$ *is used. During the **Undo** operation, the value of* X *is set to:*
A. V_1
B. V_2
C. $V_1 + V_2$
D. $V_2 - V_1$

Exercise 260 *What is the effect of the **Isolation** level "Serializable" in SQL?*
A. It uses predicate locking to prevent Phantoms.
B. It allows Non-repeatable reads but prevents Dirty reads.
C. It is implemented using only Read-Committed locks.
D. It allows multiple transactions to write to the same row simultaneously.

Exercise 261 *Consider* $S : R_1(X), W_1(X), R_2(X), W_2(X), R_3(X), W_3(X)$.
How many conflict-equivalent serial schedules exist for S*?*
A. 1
B. 3
C. 6
D. 0

Exercise 262 *If a transaction* T_i *has been granted an **Intent-Shared (IS)** lock on a table, it means* T_i*:*
A. Intends to lock the entire table in Shared mode.
B. Intends to lock individual rows of the table in Shared mode.
C. Has already locked all rows in Exclusive mode.
D. Is waiting for an Exclusive lock on the table.

Exercise 263 *Which of the following is **NOT** a method for Deadlock Handling?*
A. Wait-for Graph.
B. Thomas Write Rule.
C. Timeouts.
D. Wait-Die Scheme.

Exercise 264 *Consider* $S : R_1(X), W_2(X), W_1(X), C_1, C_2$.
Is this schedule View Serializable?
A. Yes, equivalent to $T_1 \to T_2$.
B. Yes, equivalent to $T_2 \to T_1$.
C. No, because T_1 *reads* X *before* T_2 *writes it.*
D. No, because T_2 *is the final writer but* T_1 *commits first.*

Exercise 265 *In the **Wait-Die** protocol, if an older transaction* T_{old} *requests a lock held by a younger transaction* T_{young}*:*
A. T_{old} *dies.*
B. T_{old} *waits.*
C. T_{young} *is wounded.*
D. T_{young} *is aborted.*

Exercise 266 *A schedule* S *is said to be **Conflict Serializable** if:*
A. Its precedence graph is a Directed Acyclic Graph (DAG).
B. It is view equivalent to some serial schedule.
C. It does not contain any blind writes.
D. All transactions commit at the same time.

Exercise 267 *Consider the schedule* $S : R_1(X), W_1(X), R_2(Y), W_2(Y), R_1(Y), W_1(Y)$.
The cycle in the precedence graph is:
A. $T_1 \to T_2$.
B. $T_2 \to T_1$.
C. $T_1 \to T_2 \to T_1$.
D. There is no cycle.

Exercise 268 *Which ACID property is primarily the responsibility of the **Concurrency Control Manager**?*
A. Atomicity.
B. Consistency.
C. Isolation.
D. Durability.

Exercise 269 *In a system with **Deferred Update** (No-Undo/Redo) recovery, when a transaction commits:*
A. All its updates are immediately written to the disk.
B. Its updates are recorded in the log and applied to the database.
C. The system performs an Undo of all previous transactions.
D. The log is deleted to save space.

Exercise 270 *In a system using the **Timestamp Ordering Protocol**, let* $TS(T_1) = 50$ *and* $TS(T_2) = 60$*. The following sequence occurs:* $R_1(X), W_2(X), R_1(X)$*. What is the result of the final* $R_1(X)$ *operation?*
A. It is executed normally.
B. T_1 *is rolled back because* $TS(T_1) < W\_TS(X)$.
C. It is ignored using the Thomas Write Rule.
D. T_2 *is rolled back to allow* T_1 *to read.*

Exercise 271 *Which of the following scenarios describes a **Write-Skew** anomaly, often found in Snapshot Isolation?*
A. T_1 *reads* X, T_2 *reads* X, T_1 *writes* X, T_2 *writes* X.
B. T_1 *reads* X *and* Y, T_2 *reads* X *and* Y, T_1 *writes* X, T_2 *writes* Y.
C. T_1 *writes* X, T_2 *reads* X *before* T_1 *commits.*
D. T_1 *reads* X, T_2 *deletes* X, T_1 *reads* X *again.*

Exercise 272 *Consider the schedule* $S : W_1(X), R_2(X), W_3(X), C_1, C_3, C_2$. *Which of the following serial schedules is view equivalent to* S*?*
A. $T_1 \to T_2 \to T_3$
B. $T_1 \to T_3 \to T_2$
C. $T_3 \to T_2 \to T_1$
D. No serial schedule is view equivalent.

Exercise 273 *In **Two-Phase Locking (2PL)**, the "Lock Point" is defined as:*
A. The moment the transaction begins execution.
B. The moment the transaction acquires its final lock.
C. The moment the transaction releases its first lock.
D. The moment the transaction commits.

Exercise 274 *A schedule* S *involves transactions* T_1, T_2, T_3. *The precedence graph has edges* $T_1 \to T_2$ *and* $T_2 \to T_3$. *If we add the operation* $W_3(X)$ *followed by* $R_1(X)$, *the resulting schedule:*
A. Remains conflict serializable.
B. Develops a cycle $T_1 \to T_2 \to T_3 \to T_1$.
C. Becomes view serializable only.
D. Violates the ACID property of Atomicity.

Exercise 275 *Consider the following log at the time of a system crash:*

$$\langle T_1, \textit{start} \rangle,$$
$$\langle T_1, X, 100, 200 \rangle,$$
$$\langle T_2, \textit{start} \rangle,$$
$$\langle T_2, Y, 50, 100 \rangle,$$
$$\langle \textit{checkpoint}\{T_1, T_2\} \rangle,$$
$$\langle T_3, \textit{start} \rangle,$$
$$\langle T_3, Z, 200, 300 \rangle,$$
$$\langle T_2, \textit{commit} \rangle$$

*Which transactions must be **Redone** during recovery?*
A. T_1, T_2, T_3
B. T_2 *only.*
C. T_1, T_2 *only.*
D. T_3 *only.*

Exercise 276 *In the **Tree Protocol** for graph-based locking, which of the following is TRUE?*
A. It guarantees conflict serializability and freedom from deadlock.
B. It requires all locks to be held until commit.
C. A transaction can unlock a node and then lock its child.
D. It is a variation of Strict 2PL.

Exercise 277 *Given* $S : R_1(X), R_2(Y), W_1(Y), W_2(X)$. *This schedule is an example of:*
A. A conflict serializable schedule.
B. A deadlock-prone schedule under 2PL.
C. A recoverable schedule.
D. A view serializable schedule.

Exercise 278 *A transaction T_1 reads a data item X that was updated by T_2. Before T_1 commits, T_2 aborts. T_1 then has to be aborted. This phenomenon is known as:*
A. Lost Update.
B. Dirty Read.
C. Cascading Rollback.
D. Phantom Read.

Exercise 279 *In **Rigorous 2PL**, the throughput of the system is generally:*
A. Higher than Basic 2PL due to less waiting.
B. Lower than Basic 2PL because locks are held longer.
C. Higher than Basic 2PL because it prevents deadlocks.
D. The same as Basic 2PL.

Exercise 280 *Consider the schedule $S : W_1(X), W_2(Y), W_1(Y), C_1, C_2$. The equivalent serial order is:*
A. $T_1 \rightarrow T_2$
B. $T_2 \rightarrow T_1$
C. Both A and B.
D. No serial order exists.

Exercise 281 *Which of the following properties of a schedule is **not** concerned with the order of read and write operations, but rather with the failure of transactions?*
A. Serializability.
B. Recoverability.
C. Atomicity.
D. Precedence.

Exercise 282 *In a database using **Immediate Update** recovery, the log record $\langle T_i, X, V_1, V_2 \rangle$ is used. During the **Undo** operation, the value of X is set to:*
A. V_1
B. V_2
C. $V_1 + V_2$
D. $V_2 - V_1$

Exercise 283 *What is the effect of the **Isolation** level "Serializable" in SQL?*
A. It uses predicate locking to prevent Phantoms.
B. It allows Non-repeatable reads but prevents Dirty reads.
C. It is implemented using only Read-Committed locks.
D. It allows multiple transactions to write to the same row simultaneously.

Exercise 284 *Consider $S : R_1(X), W_1(X), R_2(X), W_2(X), R_3(X), W_3(X)$. How many conflict-equivalent serial schedules exist for S?*
A. 1
B. 3
C. 6
D. 0

Exercise 285 *If a transaction T_i has been granted an **Intent-Shared (IS)** lock on a table, it means T_i:*
A. Intends to lock the entire table in Shared mode.
B. Intends to lock individual rows of the table in Shared mode.
C. Has already locked all rows in Exclusive mode.
D. Is waiting for an Exclusive lock on the table.

Exercise 286 *Which of the following is **NOT** a method for Deadlock Handling?*
A. Wait-for Graph.
B. Thomas Write Rule.
C. Timeouts.
D. Wait-Die Scheme.

Exercise 287 *Consider $S : R_1(X), W_2(X), W_1(X), C_1, C_2$. Is this schedule View Serializable?*
A. Yes, equivalent to $T_1 \rightarrow T_2$.
B. Yes, equivalent to $T_2 \rightarrow T_1$.
C. No, because T_1 reads X before T_2 writes it.
D. No, because T_2 is the final writer but T_1 commits first.

Exercise 288 *In the **Wait-Die** protocol, if an older transaction T_{old} requests a lock held by a younger transaction T_{young}:*
A. T_{old} dies.
B. T_{old} waits.
C. T_{young} is wounded.
D. T_{young} is aborted.

Exercise 289 *A schedule S is said to be **Conflict Serializable** if:*
A. Its precedence graph is a Directed Acyclic Graph (DAG).
B. It is view equivalent to some serial schedule.
C. It does not contain any blind writes.
D. All transactions commit at the same time.

Exercise 290 *Consider the schedule $S : R_1(X), W_1(X), R_2(Y), W_2(Y), R_1(Y), W_1(Y)$. The cycle in the precedence graph is:*
A. $T_1 \rightarrow T_2$.
B. $T_2 \rightarrow T_1$.
C. $T_1 \rightarrow T_2 \rightarrow T_1$.
D. There is no cycle.

Exercise 291 *Which ACID property is primarily the responsibility of the **Concurrency Control Manager**?*
A. Atomicity.
B. Consistency.
C. Isolation.
D. Durability.

Exercise 292 *In a system with **Deferred Update** (No-Undo/Redo) recovery, when a transaction commits:*
A. All its updates are immediately written to the disk.
B. Its updates are recorded in the log and applied to the database.
C. The system performs an Undo of all previous transactions.
D. The log is deleted to save space.

Exercise 293 ***[NAT]*** *Transactions T_1 (TS=10), T_2 (TS=20), T_3 (TS=30) execute the following schedule. Initially all timestamp values are 0.*

Step	**Operation**
1	$R_2(X)$
2	$R_1(X)$
3	$W_3(X)$
4	$W_1(X)$
5	$R_3(Y)$
6	$W_2(Y)$
7	$R_1(Y)$

*Apply the **Basic Timestamp Ordering** protocol step by step (a rolled-back transaction takes no further part; skip its remaining operations). How many operations trigger a **rollback**?*

Exercise 294 ***[MSQ]*** *Which of the following statements about the **Basic Timestamp Ordering** protocol are **correct**?* (Select all that apply.)
*A. The protocol is completely **deadlock-free** because transactions never wait — they either proceed or are rolled back.*
*B. The protocol guarantees **conflict serializability**; the equivalent serial order is the timestamp order of the transactions.*

*C. Schedules produced by basic Timestamp Ordering are always **recoverable**.*
*D. A transaction can suffer **starvation** if it is repeatedly rolled back and restarted with a new (larger) timestamp, always losing to later transactions.*

Exercise 295 ***[MSQ]*** *Under the **Thomas Write Rule**, which of the following are **true**?* (Select all that apply.)
*A. The Thomas Write Rule produces schedules that are **view serializable** but not necessarily conflict serializable.*
B. When $TS(T_i) < W\text{-}TS(X)$ and $TS(T_i) < R\text{-}TS(X)$, the write is ignored and T_i continues without rollback.
*C. When $TS(T_i) < R\text{-}TS(X)$, the Thomas Write Rule **still rolls back** T_i (the rule does not help in this case).*
D. The Thomas Write Rule improves concurrency over basic Timestamp Ordering by eliminating some unnecessary rollbacks.

Exercise 296 ***[NAT]*** *Transactions T_1 (TS=5), T_2 (TS=15), T_3 (TS=25). Initially $R\text{-}TS(X) = 0$, $W\text{-}TS(X) = 0$, $R\text{-}TS(Y) = 0$, $W\text{-}TS(Y) = 0$.*

Step	*Operation*
1	$W_3(X)$
2	$W_2(X)$
3	$W_1(X)$
4	$R_2(Y)$
5	$W_3(Y)$
6	$W_1(Y)$
7	$R_3(X)$

*Apply the **Thomas Write Rule** (for writes) and standard Basic TO rules (for reads). Count the total number of operations that result in a **rollback** (not counting ignored/skipped writes as rollbacks).*

Exercise 297 ***[MCQ]*** *A rolled-back transaction is restarted with its **original** timestamp in both Wait-Die and Wound-Wait. Why is this important?*
A. It prevents the restarted transaction from ever acquiring any locks.
B. It ensures the restarted transaction is treated as older*, reducing the likelihood of it being rolled back again and preventing starvation.*
C. It ensures the restarted transaction is treated as younger*, so it always waits.*
D. It resets the transaction's read set, preventing dirty reads.

Exercise 298 ***[MSQ]*** *Consider a scenario with three transactions T_1 (TS=1), T_2 (TS=2), T_3 (TS=3). T_2 holds lock on A; T_3 holds lock on B. T_1 requests lock on A (held by T_2); T_2 requests lock on B (held by T_3). Under **Wait-Die**, which of the following correctly describes what happens?*
A. T_1 waits for T_2 (since T_1 is older than T_2).
B. T_2 waits for T_3 (since T_2 is older than T_3).
C. A deadlock still occurs despite using Wait-Die.
D. No deadlock occurs.

— exercise DD-9 —

Exercise 299 ***[MCQ]*** *Which of the following **correctly** distinguishes the **preemption** behaviour of Wait-Die vs. Wound-Wait?*
A. Wait-Die is preemptive (can roll back the lock holder); Wound-Wait is non-preemptive.
B. Wound-Wait is preemptive (older requester rolls back younger holder); Wait-Die is non-preemptive (requester either waits or rolls itself back).
C. Both are preemptive; the difference is only in which transaction is chosen as victim.
D. Both are non-preemptive; they differ only in wait ordering.

Exercise 300 ***[NAT]*** *In the **Wound-Wait** scheme, transactions T_1 (TS=5), T_2 (TS=15), T_3 (TS=25) have the following simultaneous lock requests:*

- *T_2 requests a lock held by T_1.*
- *T_3 requests a lock held by T_2.*
- *T_1 requests a lock held by T_3.*

*How many transactions are **wounded (rolled back)** when all three requests are processed?*

Exercise 301 ***[MSQ]*** *Which of the following are methods of **deadlock detection** (as opposed to deadlock* prevention*)?*
*A. Maintaining and periodically checking a **Wait-For Graph (WFG)** for cycles.*
*B. Using the **Wait-Die** timestamp scheme when a lock conflict occurs.*
*C. Setting a **lock timeout**: if a transaction waits longer than a threshold, it is rolled back.*
*D. Using the **Wound-Wait** timestamp scheme when a lock conflict occurs.*

Exercise 302 ***[MCQ]*** *In the **Wait-For Graph (WFG)**, a directed edge $T_i \to T_j$ means:*
A. T_i has committed and is waiting for T_j's results.
B. T_i is waiting for T_j to release a lock.
C. T_i has wounded T_j and is waiting for T_j to roll back.
D. T_i was rolled back by T_j.

Exercise 303 ***[NAT]*** *A Wait-For Graph has 5 transactions and the following edges: $T_1 \to T_2$, $T_2 \to T_3$, $T_3 \to T_4$, $T_4 \to T_2$, $T_3 \to T_5$. How many transactions are involved in the **deadlock cycle** (the smallest cycle in the WFG)?*

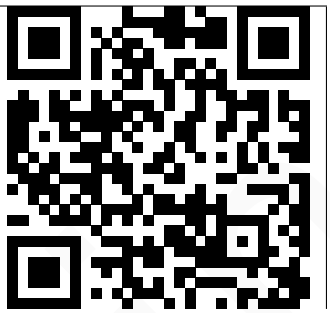
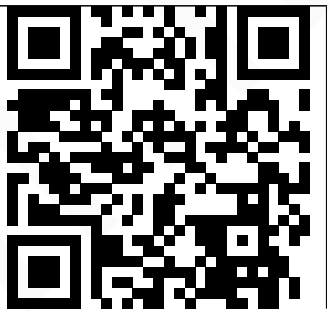
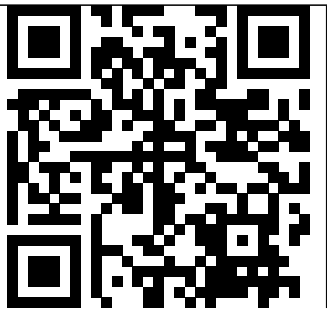
Problem 328 ***[MCQ]*** *Consider the log below for a system using **REDO-only** (no-steal/no-force) logging:*

LSN	***Log Record***	
1	$\langle T_1, \textit{begin} \rangle$	
2	$\langle T_1, X, 10 \rangle$	*(new value = 10)*
3	$\langle T_2, \textit{begin} \rangle$	
4	$\langle T_2, Y, 20 \rangle$	*(new value = 20)*
5	$\langle T_1, \textit{commit} \rangle$	
	— CRASH —	

During recovery, which operations are performed?
A. REDO T_1 (write $X = 10$); UNDO T_2 (not applicable in REDO-only — T_2 simply not replayed).
B. UNDO T_1 and UNDO T_2.
C. REDO T_1 (write $X = 10$) only; T_2's changes are discarded (never on disk).
D. REDO both T_1 (write $X = 10$) and T_2 (write $Y = 20$).

8.8 YouTube Links and QR Codes

Lecture	Details	YouTube Link	QR Code
54	Transaction Management Basics — ACID Properties & Schedules	https://youtu.be/6zSAgG5_5V4	
55	Conflict Serializability — Precedence Graph	https://youtu.be/Sp3XY5_KUIg	
56	View Serializability — View Equivalent Schedules	https://youtu.be/yBGkS0dccsE	
57	Non-Serializable Schedules — Recoverable, Cascading, Cascadeless & Strict	https://youtu.be/oYEclhA_I-8	

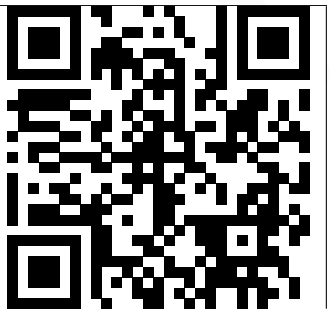
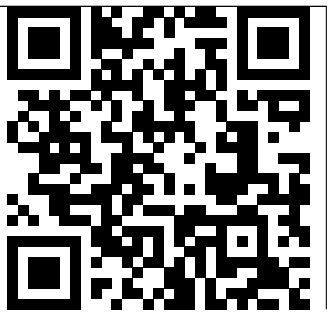
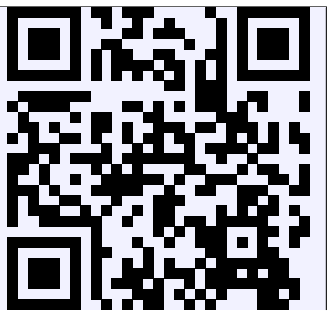
58	Concurrency Control — Locking Protocols & 2PL Variants	https://youtu.be/62rEkuFOlng	
59	Multiple Granularity Locking — Intention Locks & Lock Hierarchy	https://youtu.be/uj-TJub8D_M	
60	Graph & Tree-Based Locking Protocols	https://youtu.be/mLBL2s16lqc	
61	Timestamp Ordering Protocol — Basic, Strict & Thomas Write Rule	https://youtu.be/OWM_cVpsCKg	
62	Deadlock Prevention (Wait-Die, Wound-Wait) & Recovery Logs	https://youtu.be/jiWJfiIvCcg	

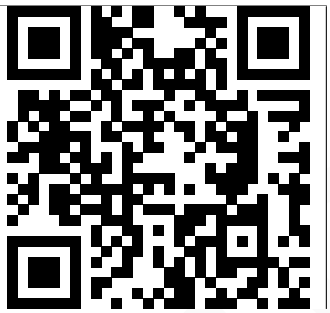
63	Practice Problems on Transactions & Concurrency Control	https://youtu.be/uNlHsbouh_I	
64	Solved GATE PYQs on Transaction Management & Concurrency Control	https://youtu.be/Yn713xjqlU4	

Chapter 9

Solutions to Practice Problems

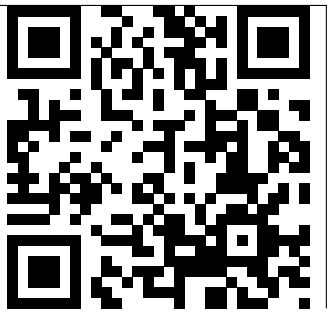
Problems Covered	YouTube Link	QR Code
Problems 1–33 (Lecture 4)	https://youtu.be/fR2goPCBLmQ	
Problems 34–64 (Lecture 9)	https://youtu.be/9PkYnKQ2-CE	
Problems 65–120 (Lecture 18)	https://youtu.be/v6tFEFkdmzw	

Problems 121–133 (Lecture 24)	https://youtu.be/X1j2eOCrrl4	
Problems 134–137 (Lecture 27)	https://youtu.be/zexCoq_YBEU	
Problems 138–181 (Lecture 34)	https://youtu.be/SBm67noqA6A	
Problems 182–231 (Lecture 46)	https://youtu.be/QqIpR3hJBuc	
Problems 232–260 (Lecture 52)	https://youtu.be/pQOsn75d3u0	

Problems 261–327 (Lecture 63)	https://youtu.be/uNlHsbouh_I	

Chapter 10

GATE PYQs Solutions

Problems Covered	YouTube Link	QR Code
PB 1-7: GATE PYQs on ER Diagrams	https://youtu.be/5ciP-BzCCuQ	
PB 8-20: Solved GATE PYQs on Relational Model	https://youtu.be/xSbc4yenX-Y	
Pb 21-44: GATE PYQs on Relational Algebra — DBMS	https://youtu.be/rXzzIc99B1w	

Pb 45-47: Tuple Relational Calculus (TRC) — Solved GATE PYQs & Practice Problems — DBMS	https://www.youtube.com/watch?v=zexCoq_YBEU&t=520s	
Pb 48-79: GATE PYQs on SQL	https://youtu.be/qFpTahjWSHc	
Pb 80-125: GATE PYQs on Normalization — FD, Closure, Keys, Decomposition, Normal Forms	https://youtu.be/iE93td4uU0s	
Pb 126-145: Solved GATE PYQs on Record Organization & Indexing	https://youtu.be/BDTNPLtW4nE	
Pb 146-178: Solved GATE PYQs on Transaction Management & Concurrency Control	https://youtu.be/Yn713xjqlU4	

Bibliography

[1] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed., McGraw-Hill Education, New York, 2020.

[2] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed., McGraw-Hill, New York, 2003.

[3] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed., Pearson, Boston, 2016.

[4] J. D. Ullman and J. Widom, *A First Course in Database Systems*, 3rd ed., Pearson, Boston, 2008.

[5] C. J. Date, *An Introduction to Database Systems*, 8th ed., Addison-Wesley, Boston, 2003.

GateXAIML

Free GATE resources for Data Science & AI and CSE
Website: www.gatexaiml.in
Email: contact@gatexaiml.in