Database Management System

A comprehensive course for Diploma Engineering students in Information & Communication Technology at Gujarat Technological University

COURSE OVERVIEW

Course Structure & Learning Path

01

Introduction to Database Systems

Foundation concepts, architecture, and database administrators

02

ER Model & Relational Algebra

Entity-relationship modeling and database design fundamentals

03

Structured Query Language

SQL commands, functions, and database manipulation

04

Normalization

Refining database design through normal forms

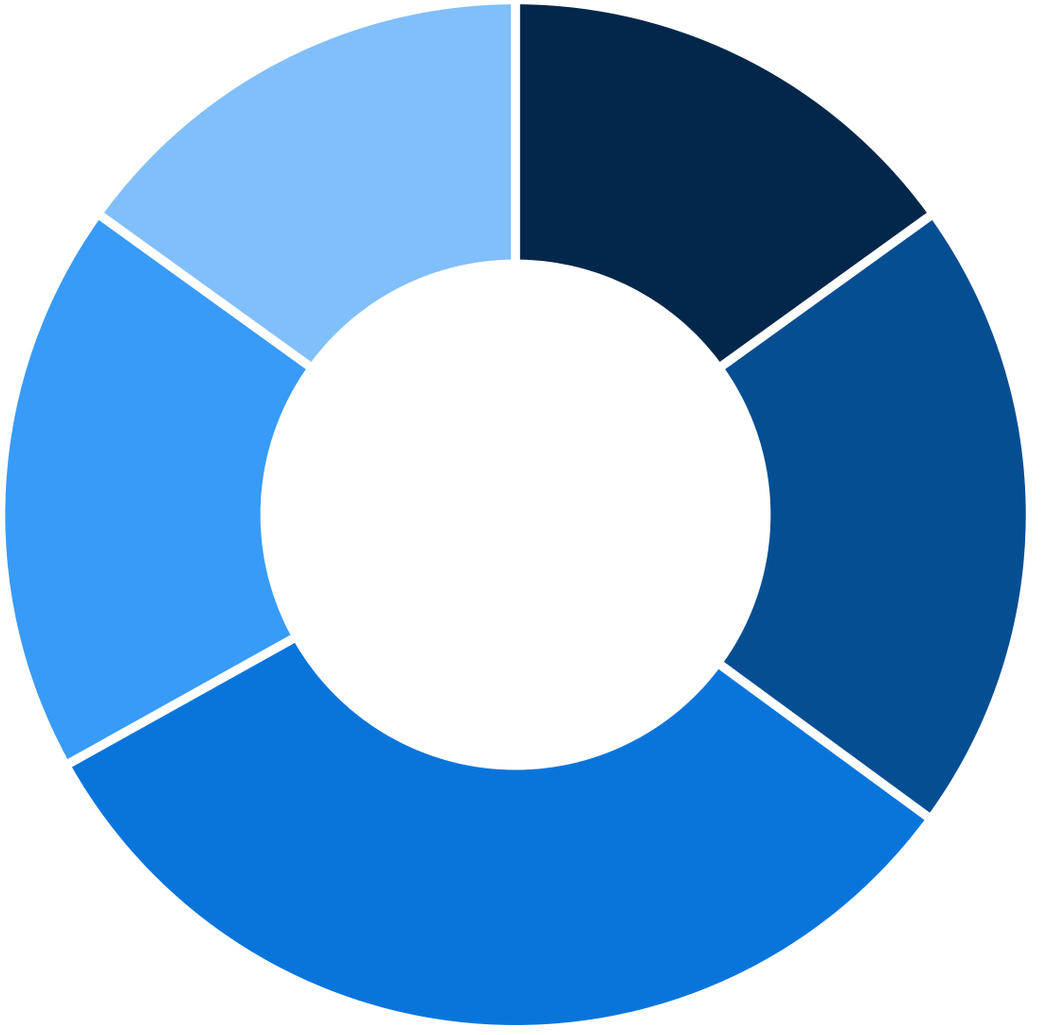
05

Transaction Management

Ensuring data consistency and integrity

UNIT WEIGHTAGE

Assessment Distribution



The course allocates 30 total hours across five units, with SQL receiving the most emphasis at 32% due to its practical importance in database management.

UNIT 1

Introduction to Database Systems

Database systems form the backbone of modern information technology, storing and managing vast amounts of data efficiently. This unit establishes foundational concepts that every ICT professional must understand, from basic terminology to complex architectural models.

Understanding these fundamentals enables students to design, implement, and manage robust database solutions for real-world applications.

Unit Details

Duration: 4 hours

Weightage: 15%

Topics: 10 major concepts

Data vs. Information: Understanding the Difference

Data

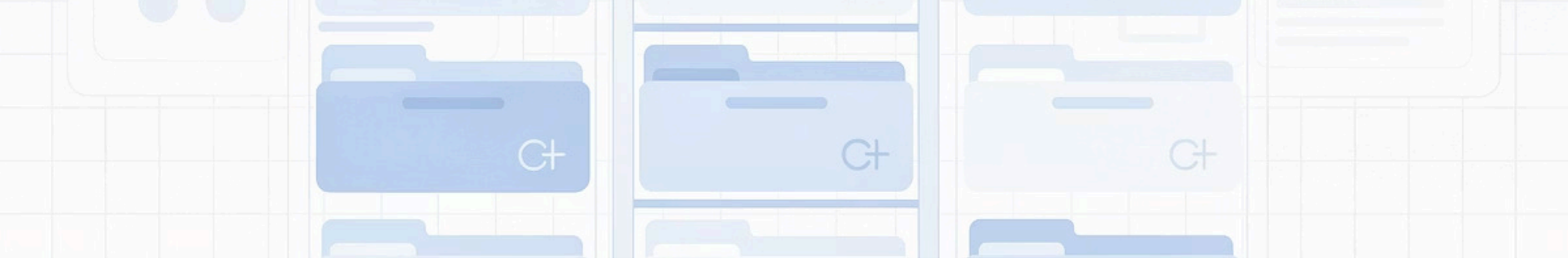
Raw, unprocessed facts and figures without context. Data represents individual observations, measurements, or records that haven't been analyzed or interpreted.

- Example: 98.6, "John Smith", 2024
- No inherent meaning alone
- Requires processing

Information

Processed, organized, and structured data that provides meaningful context. Information results from analyzing data and presenting it in a useful format.

- Example: Patient John Smith's temperature is 98.6°F on January 15, 2024
- Actionable and meaningful
- Supports decision-making



What is a Database?

A database is an organized collection of structured data stored electronically in a computer system. It enables efficient storage, retrieval, and management of information through systematic organization and relationships between data elements.

Structured Organization

Data arranged in tables, rows, and columns for easy access and manipulation

Persistent Storage

Information retained over time, surviving program executions and system restarts

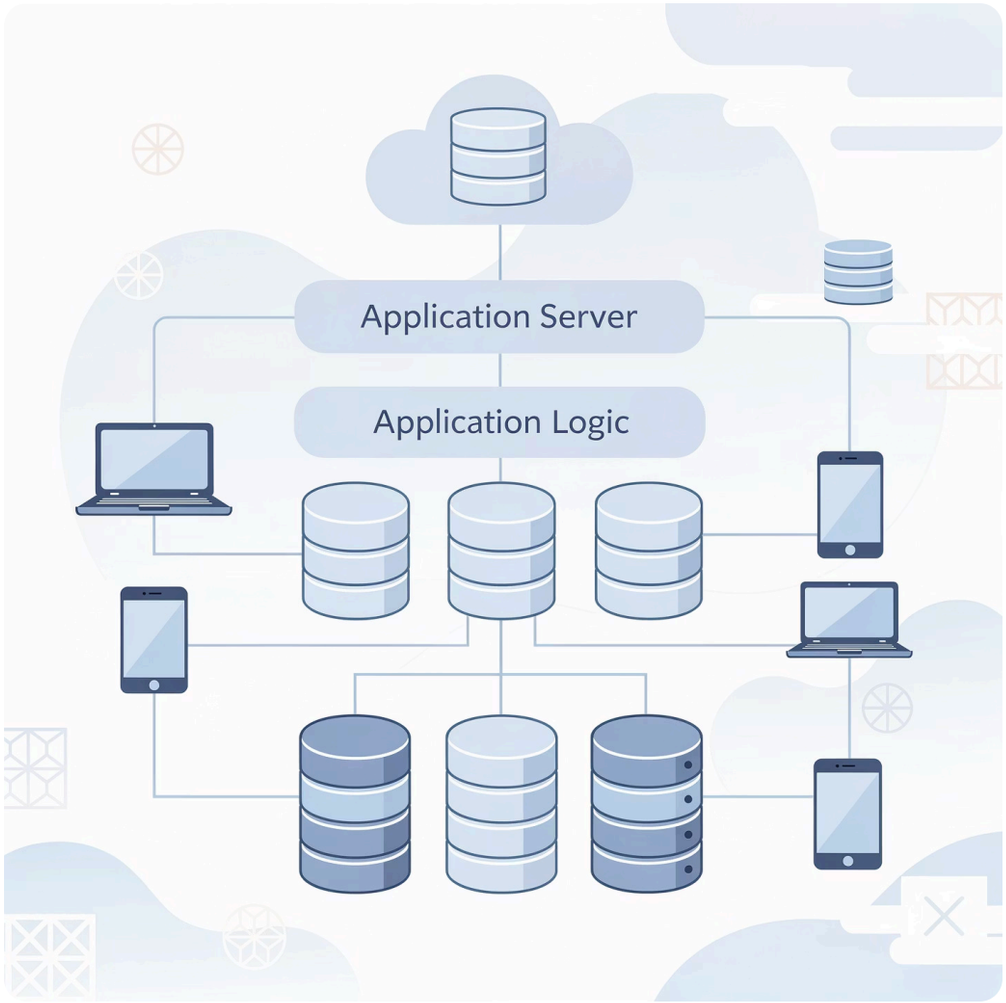
Shared Access

Multiple users can access and modify data simultaneously with proper controls

Database Management System (DBMS)

A Database Management System is sophisticated software that provides an interface between the database and users or application programs. It handles all requests for data access, ensuring security, integrity, and efficient management.

Popular DBMS examples include MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL, and MongoDB. Each offers unique features suited to different application requirements.



Security Control

Manages user authentication, authorization, and access permissions

Data Management

Handles storage, retrieval, updates, and deletion operations

Performance Optimization

Ensures fast query execution through indexing and caching

Metadata: Data About Data

> "Metadata is the librarian of the database world – it tells you where everything is and what it means."

Metadata describes the structure, organization, and characteristics of data stored in a database. It includes information about table names, column names, data types, constraints, relationships, and other properties that define how data is organized and interpreted.

Structural Metadata

- Table schemas
- Column definitions
- Data types
- Primary keys

Descriptive Metadata

- Field descriptions
- Business rules
- Valid value ranges
- Default values

Administrative Metadata

- Creation dates
- Modification history
- Access permissions
- Backup information

Database Building Blocks

Data Items

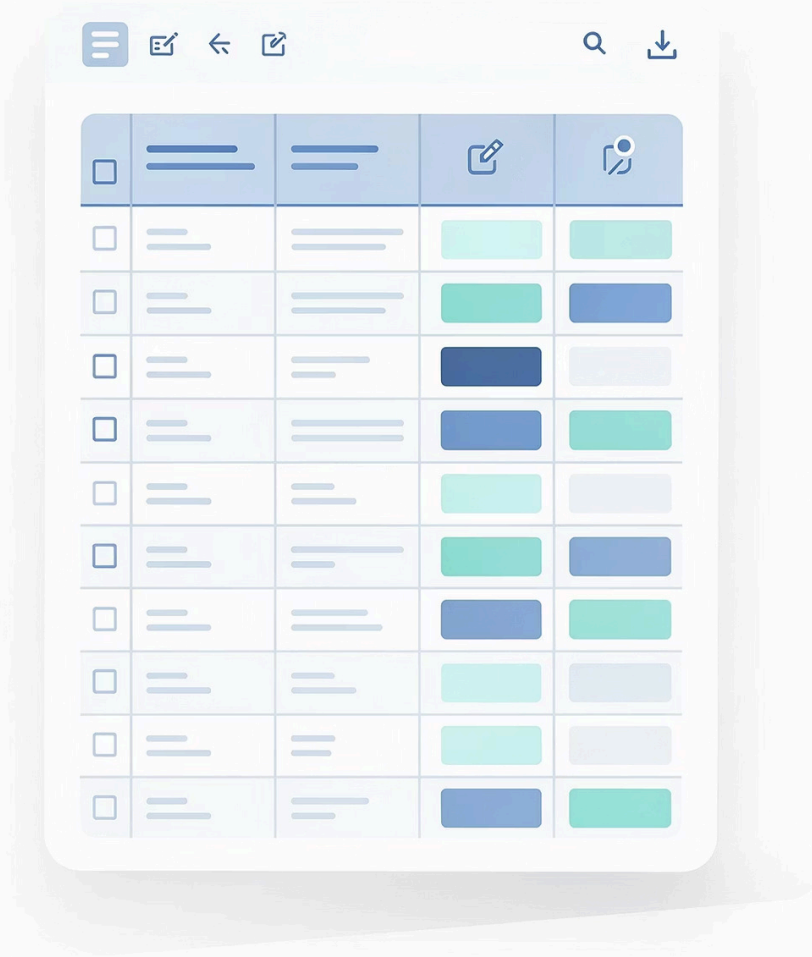
The smallest unit of data that has meaning, representing a single fact or value such as a name, number, or date.

Fields

A collection of related data items forming a column in a table, such as "Customer Name" or "Order Date".

Records

A complete set of fields representing a single entity, forming a row in a table containing all information about one customer, product, or transaction.



Data Dictionary: The Database Encyclopedia

A data dictionary is a centralized repository that stores metadata about all data elements in the database. It serves as a comprehensive reference guide documenting the structure, organization, and meaning of database components.

Table Definitions

Names, purposes, and descriptions of all tables in the database

Column Specifications

Data types, sizes, constraints, and valid values for each field

Relationship Mapping

Foreign keys, joins, and connections between different tables

Business Rules

Validation rules, calculations, and logic governing data entry

Purpose of Database Systems

Database systems solve critical data management challenges faced by organizations. They eliminate data redundancy, ensure consistency, provide concurrent access, maintain security, and enable efficient data retrieval and analysis.

Reduce Redundancy

Store data once, eliminating duplicate copies and inconsistencies

Maintain Integrity

Enforce rules ensuring data accuracy and consistency across the system

Enable Sharing

Allow multiple users to access and modify data simultaneously

Ensure Security

Control access through authentication and authorization mechanisms

File-Oriented vs. Database Systems

File-Oriented System

- Data stored in separate files
- High redundancy
- Difficult to update
- Limited data sharing
- No centralized control
- Inconsistency problems

Database System

- Centralized data storage
- Minimal redundancy
- Easy updates
- Efficient data sharing
- Centralized administration
- Guaranteed consistency

Applications of DBMS

Banking & Finance

Managing customer accounts, transactions, loans, and investment portfolios with real-time processing

Healthcare

Storing patient records, medical histories, prescriptions, and appointment scheduling systems

E-Commerce

Product catalogs, inventory management, customer orders, and payment processing systems

Education

Student records, course registration, grades, attendance tracking, and learning management

Airlines

Flight schedules, reservations, passenger information, and crew management systems

Telecommunications

Call records, billing information, network management, and customer service databases

Database Administrator (DBA)

The Database Administrator is the guardian of organizational data, responsible for the performance, integrity, and security of databases. This critical role requires technical expertise, problem-solving skills, and strategic planning abilities.

DBAs bridge the gap between business needs and technical implementation, ensuring databases support organizational goals while maintaining optimal performance and security standards.

Roles & Responsibilities of DBA

Installation & Configuration

Set up database software and configure system parameters

Documentation

Maintain comprehensive records of configurations and procedures

Monitoring & Maintenance

Track system health, apply updates, and resolve issues proactively

Security Management

Control user access, implement encryption, and protect against threats

Performance Tuning

Optimize queries, indexes, and system resources for efficiency

Backup & Recovery

Implement strategies to protect data and restore after failures

Schema, Sub-Schema, and Instances

Schema

The overall logical structure and design of the database, defining tables, fields, relationships, and constraints. It represents the blueprint of how data is organized.

Example: Student(ID, Name, Age, Major)

Sub-Schema

A subset of the schema visible to specific users or applications, providing a customized view of the database tailored to particular needs or security requirements.

Example: Faculty sees only Name and Major, not Age

Instance

The actual data stored in the database at a particular moment in time. Instances change frequently as data is inserted, updated, or deleted.

Example: Specific student records at 3:00 PM on Jan 15, 2024

Data Abstraction: Simplifying Complexity

Data abstraction hides complex implementation details from users, presenting only essential information through different levels of representation. This multi-layered approach enables different users to interact with the database at appropriate levels of detail.

External Level

User view - how end users see data

Conceptual Level

Logical view - what data is stored and relationships

Internal Level

Physical view - how data is actually stored on disk

Internal Level: Physical Storage

The internal level describes how data is physically stored on storage devices. It deals with data structures, file organizations, access methods, and storage allocation strategies.

This lowest level of abstraction focuses on optimizing storage efficiency and access performance through techniques like indexing, compression, and strategic data placement.

- **Storage Structures**

  B-trees, hash tables, and clustered indexes for organizing data on disk

- **Access Paths**

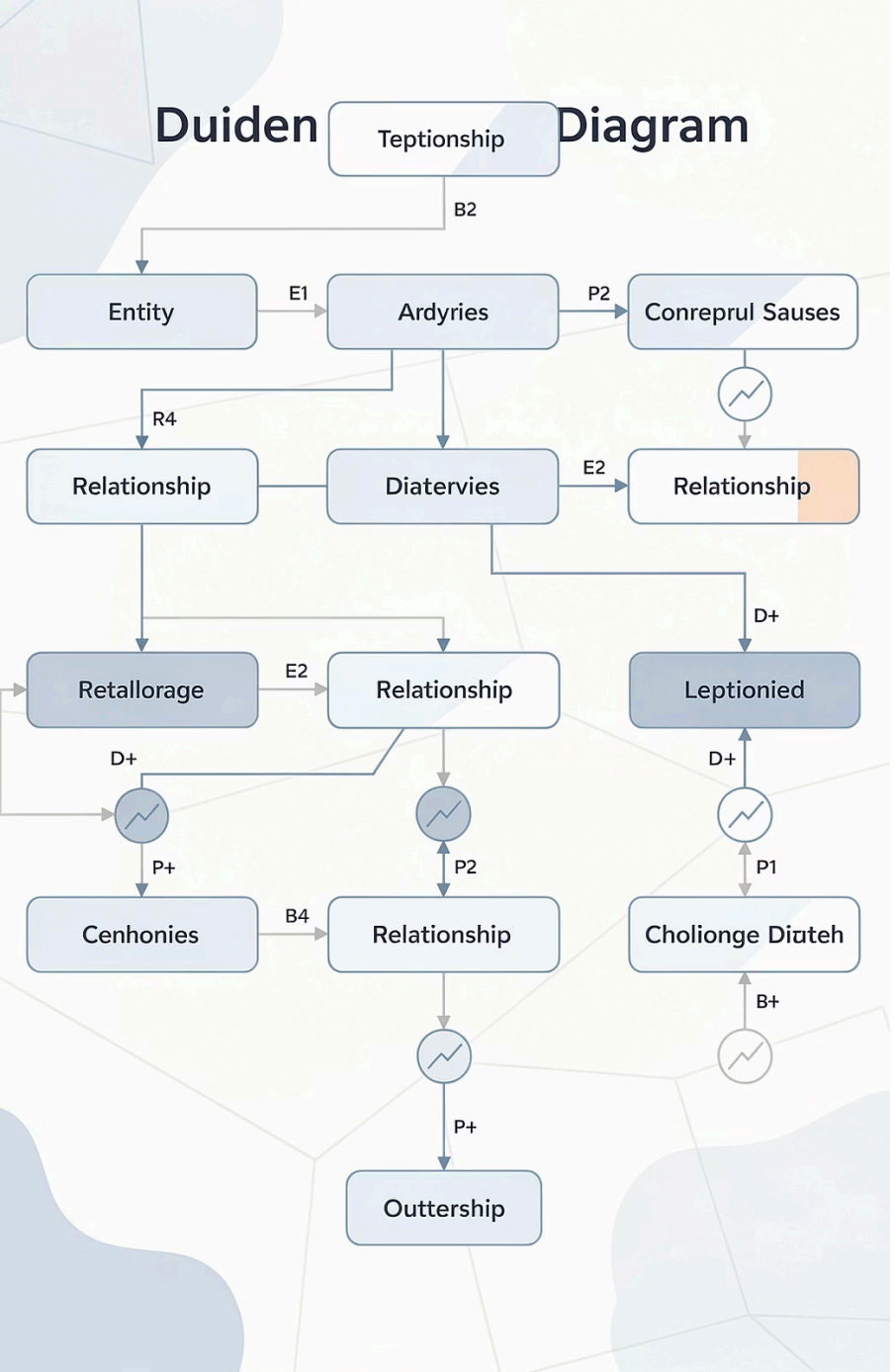
  Methods for locating and retrieving data efficiently from storage

- **Compression Techniques**

  Algorithms to reduce storage space while maintaining data integrity

- **Buffer Management**

  Memory allocation strategies for caching frequently accessed data



Conceptual Level: Logical Structure

The conceptual level defines the logical structure of the entire database for all users. It describes what data is stored, the relationships between data elements, and constraints, without concerning itself with physical storage details.

Entity Definitions

Specification of all entities (tables) in the database with their attributes and data types

Relationships

Connections between entities through primary and foreign keys defining how data relates

Constraints

Business rules and integrity constraints ensuring data validity and consistency

External Level: User Views

> Different users see different perspectives of the same database based on their roles and requirements.

Students

View personal information, course enrollments, grades, and schedules without seeing administrative data

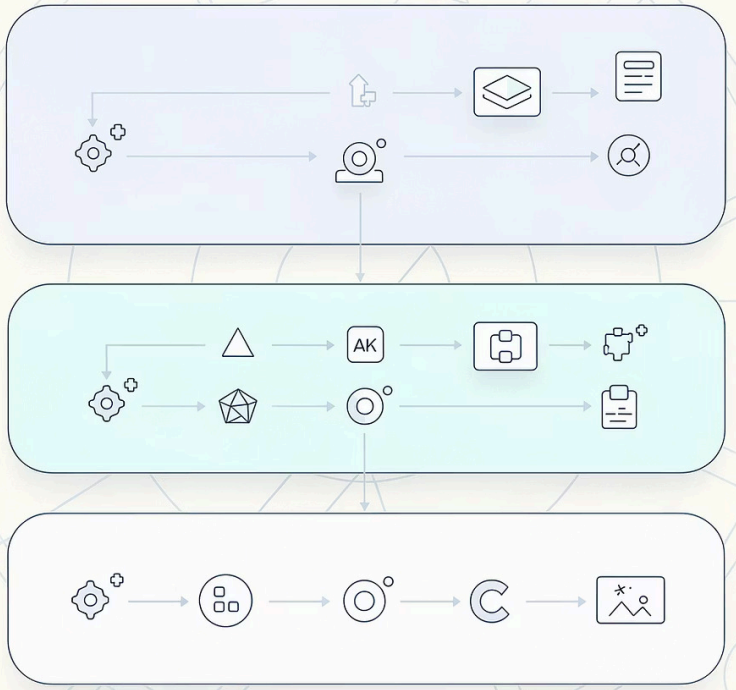
Faculty

Access course rosters, student performance, and grade submission tools without financial information

Administrators

See comprehensive data including enrollment statistics, financial records, and system-wide reports

Security Benefit: External views provide data security by limiting what each user can see and modify, implementing the principle of least privilege.



Data Independence

Data independence is the capacity to change one level of the database architecture without affecting other levels. This powerful feature allows system modifications without disrupting applications or user interactions.

Physical Independence

The ability to change physical storage structures or devices without modifying the conceptual schema or application programs.

- Change storage devices
- Modify indexing strategies
- Reorganize file structures
- Applications remain unaffected

Logical Independence

The capacity to change the conceptual schema without altering external schemas or application programs.

- Add new entities or attributes
- Modify relationships
- Change constraints
- Existing views stay functional

Database Architecture Models

Various database models provide different approaches to organizing and representing data. Each model has distinct characteristics, advantages, and ideal use cases based on the nature of data and application requirements.

1

Hierarchical Model (1960s)

Tree-like structure with parent-child relationships

2

Network Model (1970s)

Graph structure allowing multiple parent-child connections

3

Relational Model (1970s)

Table-based organization with rows and columns

4

Object-Oriented Model (1980s)

Objects with properties and methods like programming

5

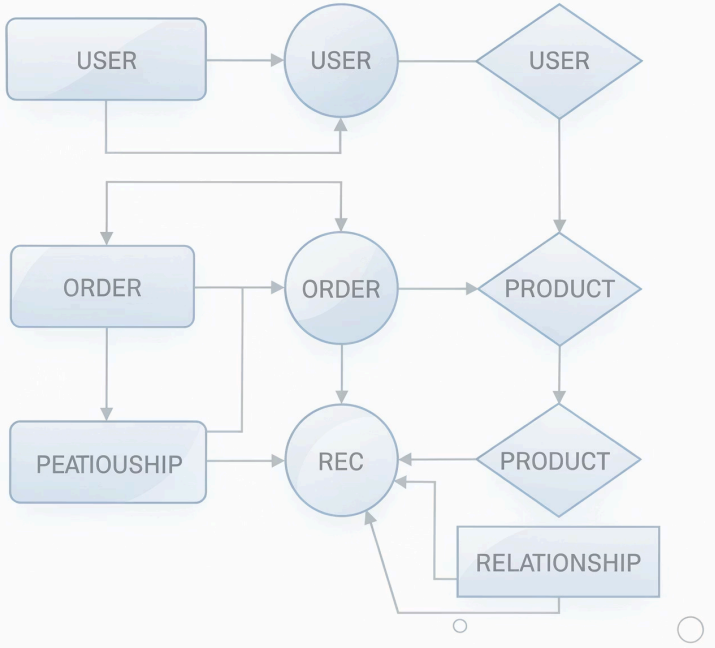
ER Model (1976)

Conceptual design using entities and relationships

Entity-Relationship (ER) Model

The ER model is a high-level conceptual data model used for designing databases. It represents real-world entities, their attributes, and relationships between entities using graphical diagrams.

ER modeling provides an intuitive way to visualize database structure before implementation, facilitating communication between database designers, developers, and stakeholders.



Entities

Real-world objects or concepts (Student, Course, Employee)

Attributes

Properties describing entities (Name, ID, Date)

Relationships

Associations between entities (Enrolls, Teaches, Works)

Relational Model

The relational model organizes data into tables (relations) consisting of rows (tuples) and columns (attributes). Introduced by E.F. Codd in 1970, it remains the most widely used database model due to its simplicity, flexibility, and mathematical foundation.

Tables

Two-dimensional structures representing entities with rows and columns

Keys

Unique identifiers establishing relationships between tables

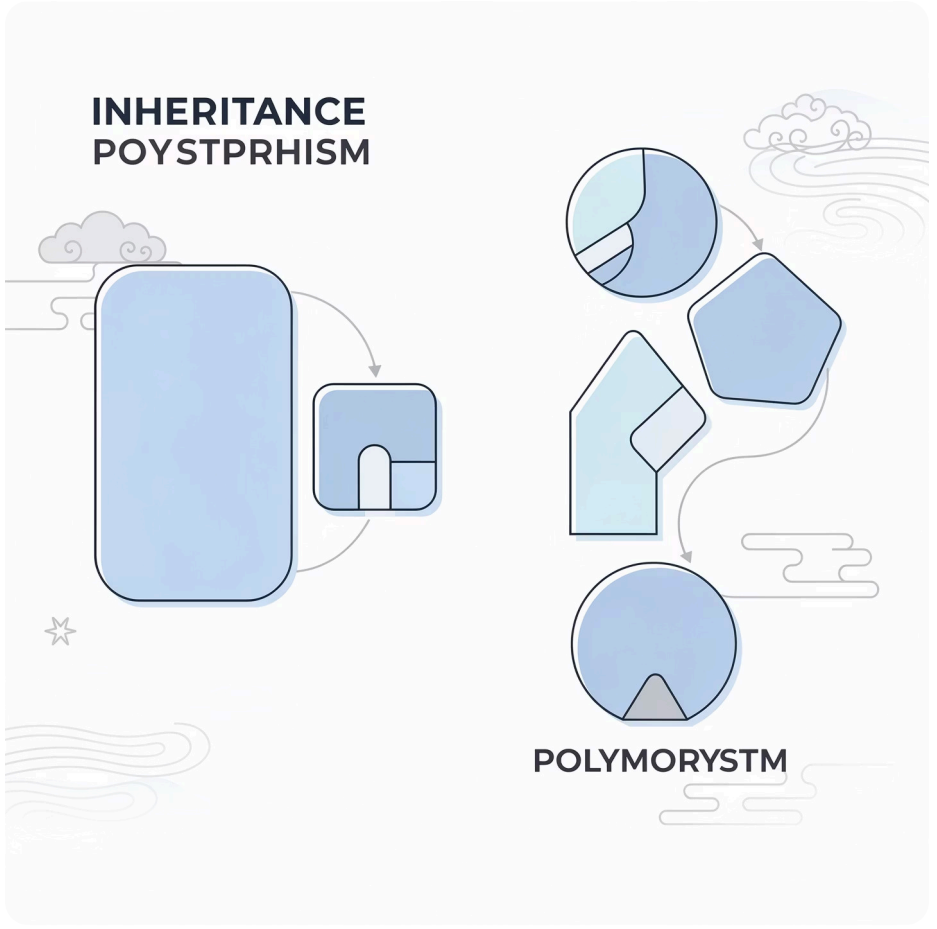
SQL

Structured Query Language for managing and querying relational databases

Normalization

Process of organizing data to reduce redundancy and improve integrity

Object-Oriented Data Model



The object-oriented model represents data as objects, similar to object-oriented programming. Objects encapsulate data and methods, supporting complex data types and relationships through inheritance and polymorphism.

This model excels at handling multimedia data, CAD systems, and applications requiring complex data structures beyond simple tables.

Encapsulation

Combining data and methods that operate on that data within objects

Inheritance

Creating new object classes based on existing ones, inheriting properties

Polymorphism

Objects of different types responding to the same method calls differently

Network Data Model

The network model extends the hierarchical model by allowing records to have multiple parent and child records, forming a graph structure. This flexibility enables representation of complex many-to-many relationships.

1

Sets

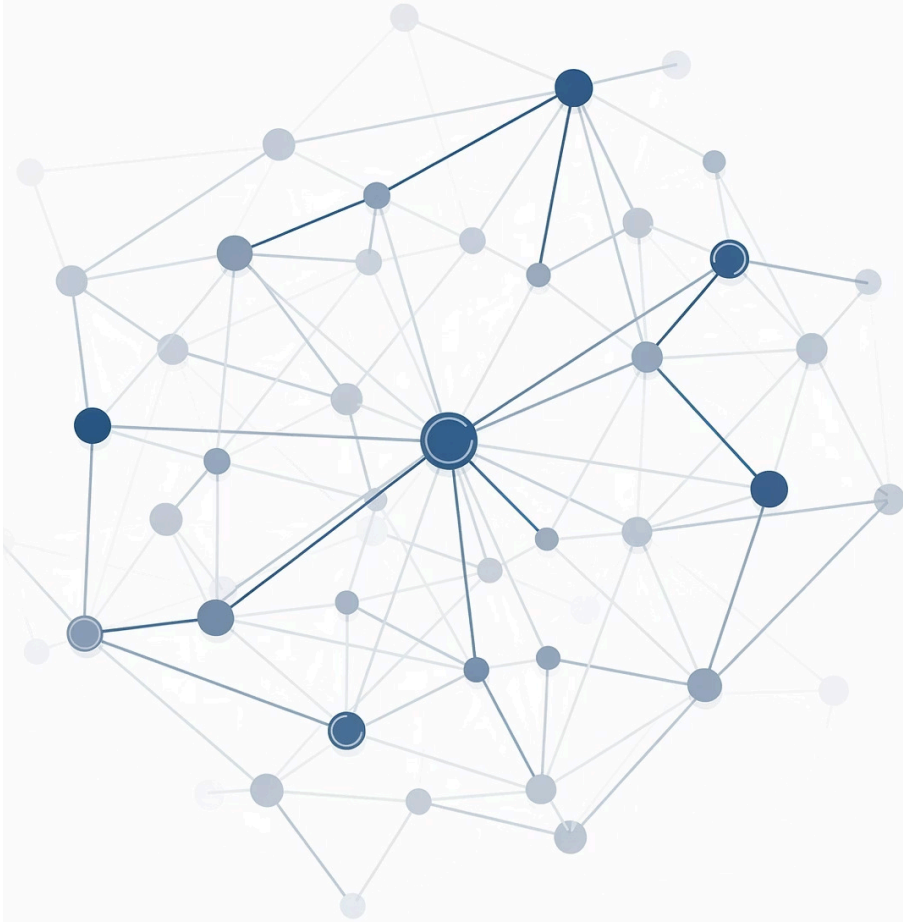
Collections of records with owner-member relationships defining connections

2

Pointers

Navigation through the database using explicit links between records

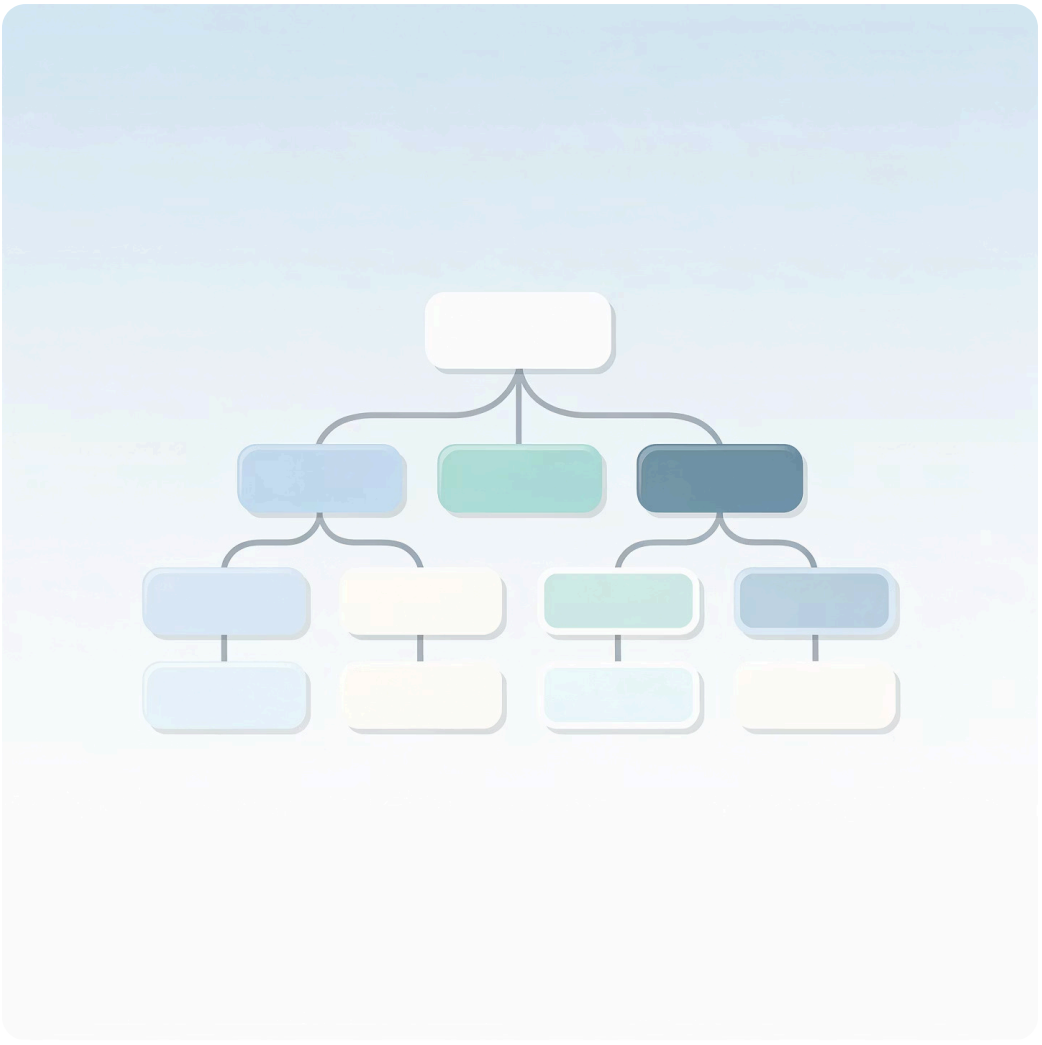
Historical Note: While largely superseded by relational databases, network databases influenced modern graph databases used for social networks and recommendation systems.

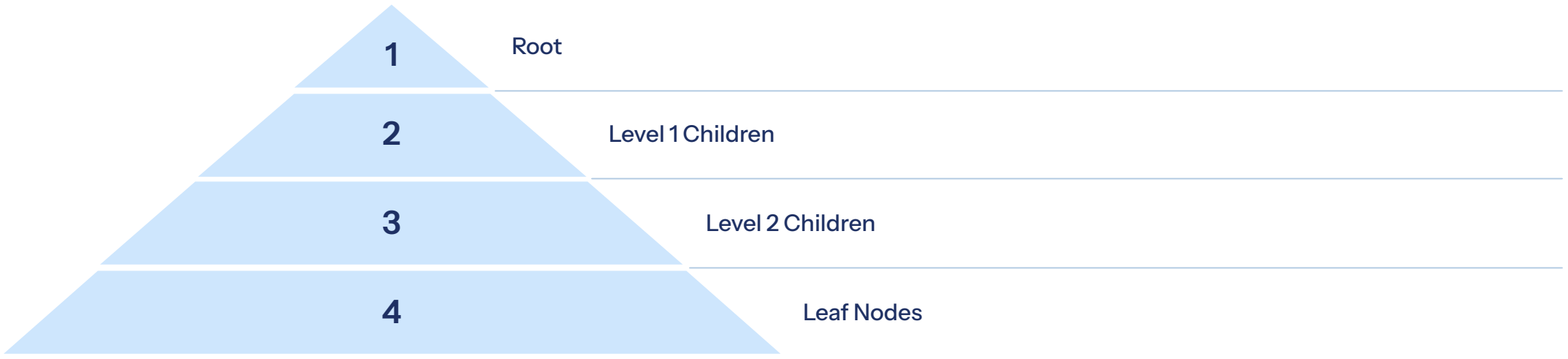


Hierarchical Data Model

The hierarchical model organizes data in a tree-like structure with parent-child relationships. Each child record has exactly one parent, forming a strict hierarchy ideal for representing organizational structures and file systems.

IBM's Information Management System (IMS) popularized this model in the 1960s, and its concepts persist in XML documents and file system organization.





Databases in Smart Cities

Modern smart cities rely heavily on database systems to collect, store, and analyze massive volumes of data from sensors, devices, and citizens. These systems enable real-time decision-making, resource optimization, and improved quality of life.

Traffic Management

Real-time traffic flow data, signal optimization, and congestion prediction

Energy Distribution

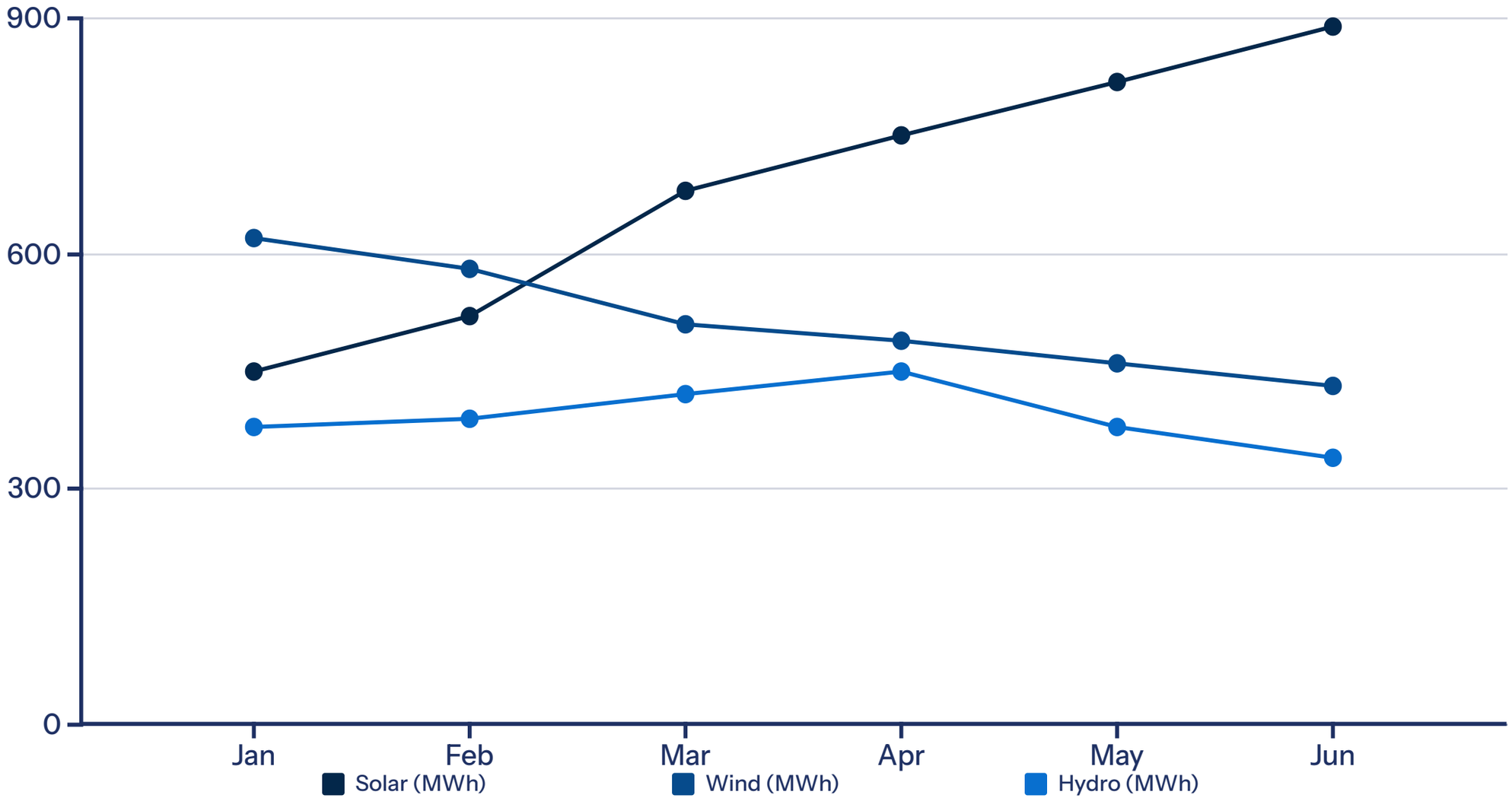
Grid monitoring, demand forecasting, and renewable integration

Public Safety

Emergency response coordination, surveillance analysis, and incident tracking

Renewable Energy Tracking

Database systems play a crucial role in monitoring and optimizing renewable energy sources. They track production metrics, consumption patterns, and environmental impact while enabling predictive maintenance and efficient energy distribution.



This data enables grid operators to balance supply and demand, integrate intermittent renewable sources, and optimize energy storage systems for maximum efficiency.

UNIT 2

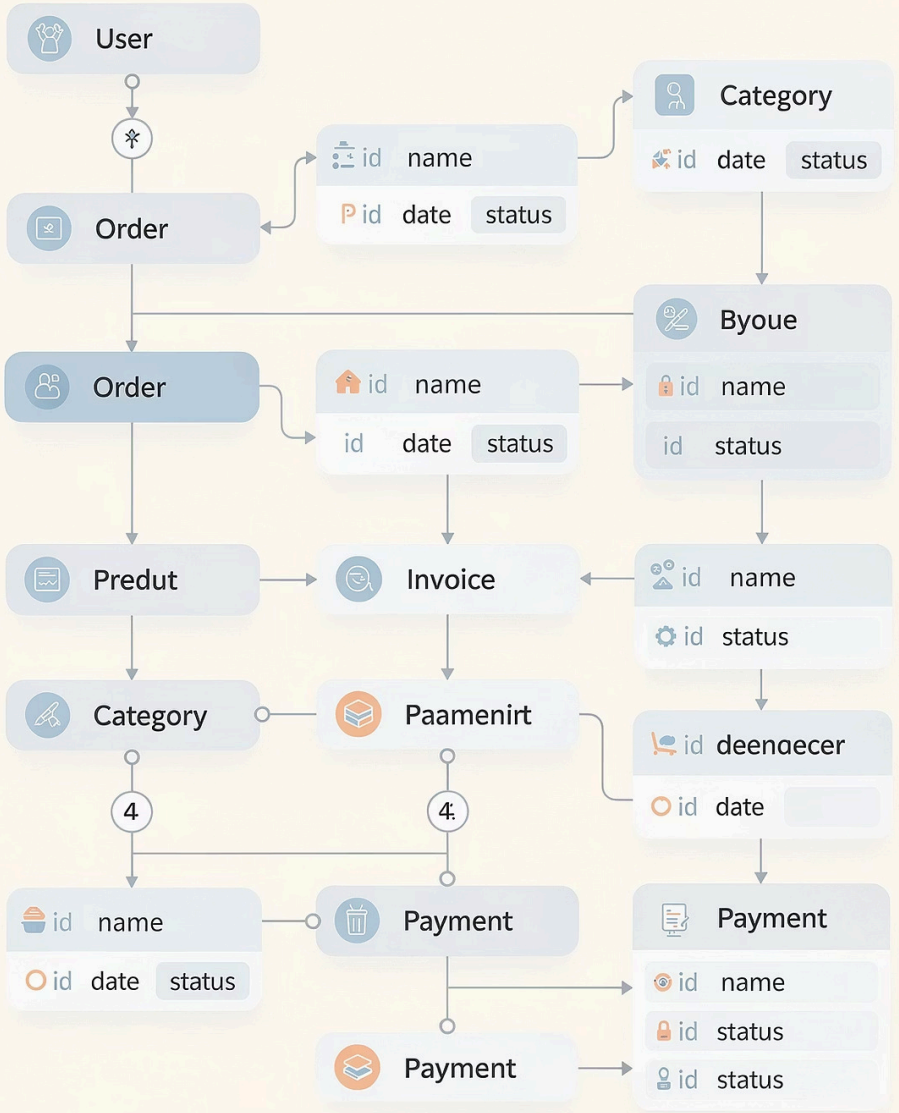
ER Model and Relational Algebra

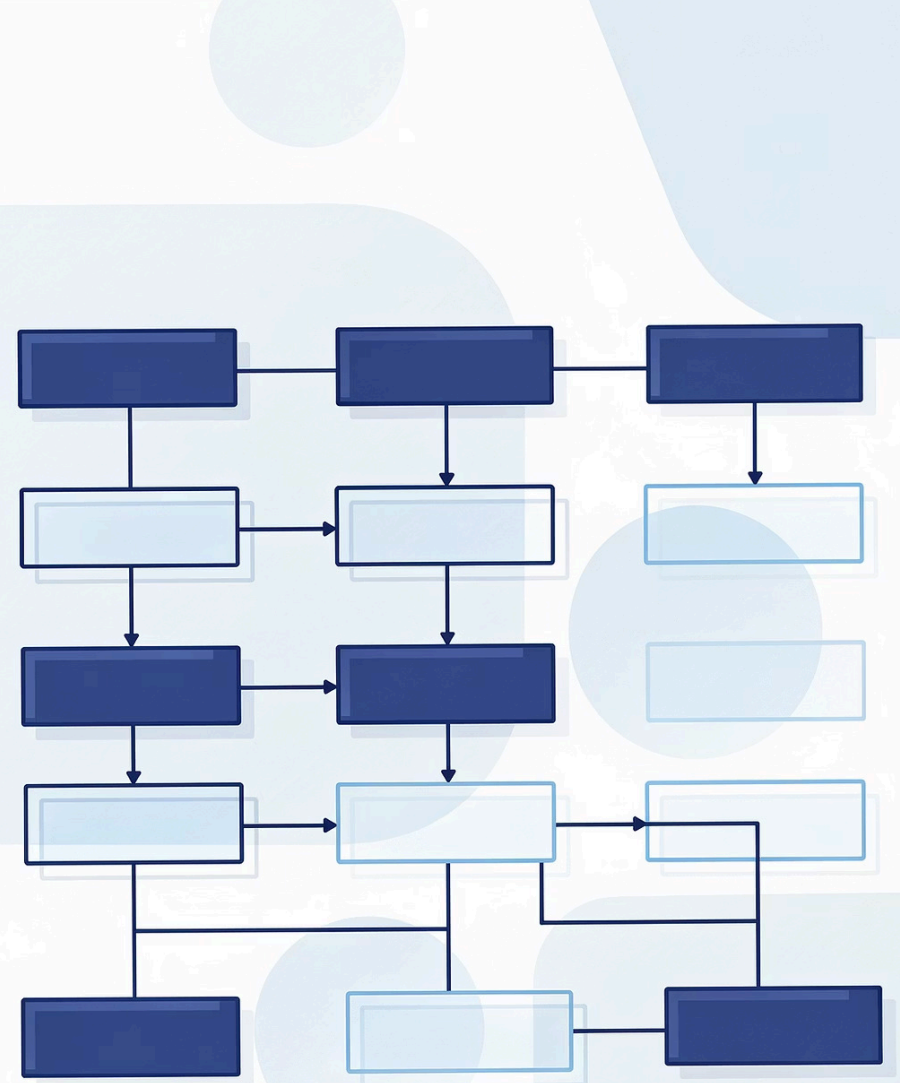
The Entity-Relationship model provides a powerful conceptual framework for database design. This unit explores how to model real-world scenarios using entities, attributes, and relationships, then convert these designs into functional database schemas.

Understanding ER modeling is essential for creating well-structured databases that accurately represent business requirements while maintaining data integrity and supporting efficient queries.

Duration: 5 hours

Weightage: 20%





Entity: The Foundation of Data Modeling

An entity represents a distinguishable real-world object or concept about which information is stored. Entities can be concrete (like a person or product) or abstract (like an event or transaction).

Strong Entity

Exists independently with its own unique identifier (primary key). Example: Student, Employee, Product

Weak Entity

Depends on another entity for identification and existence. Example: Order Line Item depends on Order

Identifying entities correctly is the first critical step in database design, establishing what information the system will track and manage.

Attributes: Describing Entities

Attributes are properties or characteristics that describe entities. Each entity has a set of attributes that capture relevant information about that entity type.

For example, a Student entity might have attributes like StudentID, Name, DateOfBirth, Email, and Major.

Simple Attributes

Atomic values that cannot be divided further (Age, Price, ISBN)

Composite Attributes

Can be divided into smaller sub-parts (Address = Street, City, State, ZIP)

Derived Attributes

Calculated from other attributes (Age derived from DateOfBirth)

Multi-valued Attributes

Can have multiple values (Phone Numbers, Email Addresses)

Relationships: Connecting Entities

Relationships represent meaningful associations between entities. They capture how entities interact, depend on each other, or are related in the real-world scenario being modeled.

Unary (Recursive)

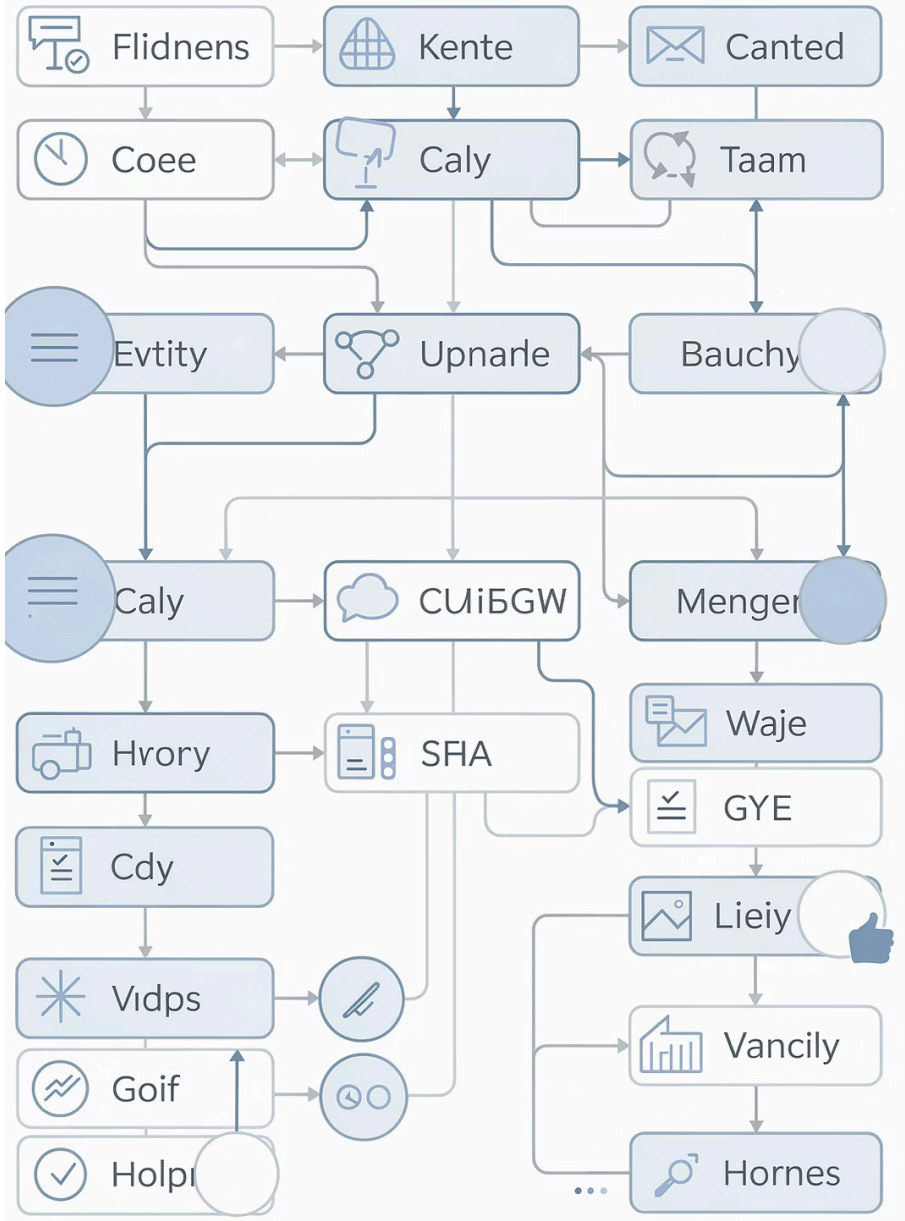
Entity related to itself (Employee manages Employee)

Binary

Between two entity types (Student enrolls in Course)

Ternary

Involving three entities (Doctor prescribes Drug to Patient)



Participation in Relationships

Total Participation

Every entity instance must participate in the relationship. Represented by double lines in ER diagrams. Example: Every Order must have an associated Customer.

This enforces business rules requiring the relationship - you cannot have an orphaned entity.

Partial Participation

Entity instances may or may not participate in the relationship. Represented by single lines. Example: Not every Employee manages a Department.

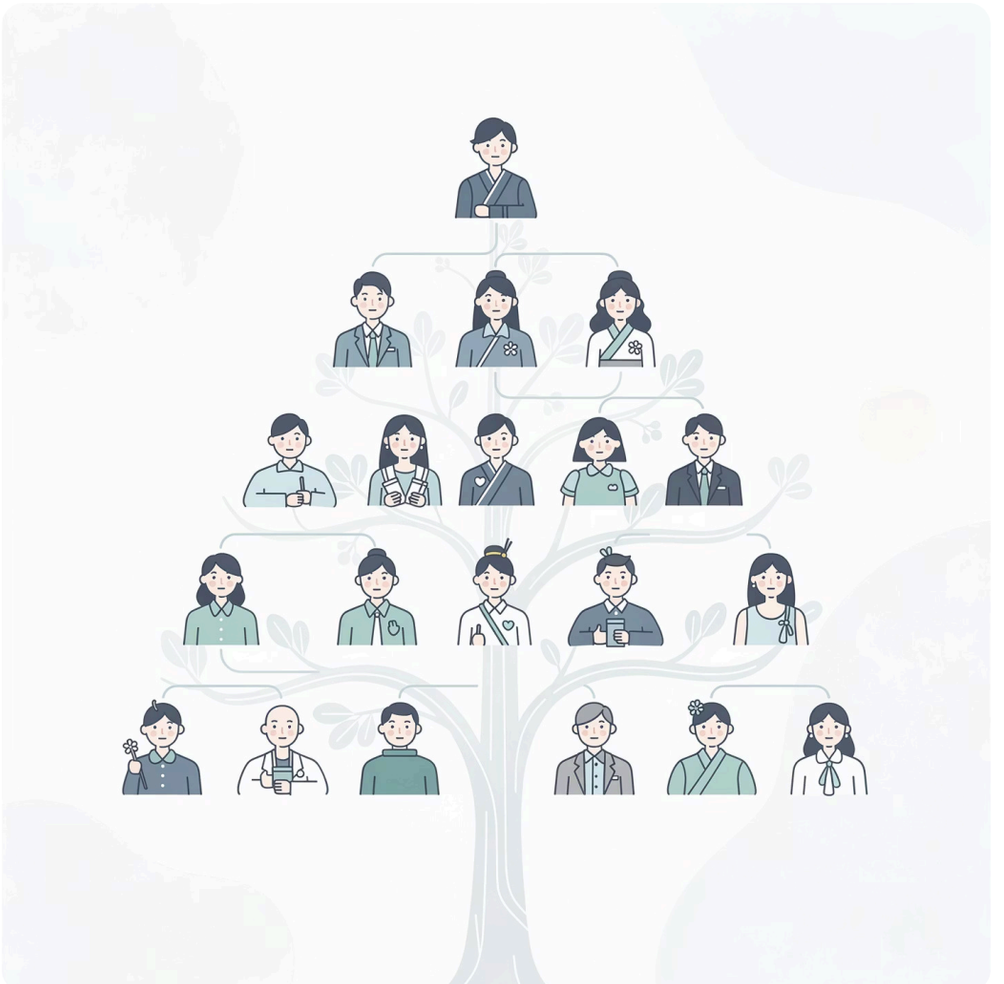
This allows optional relationships where the association is not mandatory for all instances.

Understanding participation constraints is crucial for implementing proper foreign key constraints and ensuring data integrity.

Recursive Relationships

A recursive (or unary) relationship involves a single entity type where instances relate to other instances of the same entity. These model hierarchical structures and self-referential associations.

Common examples include organizational hierarchies, social networks, and prerequisite structures in academic courses.



Management Hierarchy

Employee "manages" Employee - supervisors and subordinates within the same entity

Course Prerequisites

Course "requires" Course - courses that must be completed before others

Social Connections

Person "follows" Person - social media connections between users

Degree of Relationship Set

The degree of a relationship indicates the number of entity types participating in the relationship. Understanding degree helps in accurately modeling complex business scenarios.

Unary (Degree 1)

One entity type. Example: Employee supervises Employee

Binary (Degree 2)

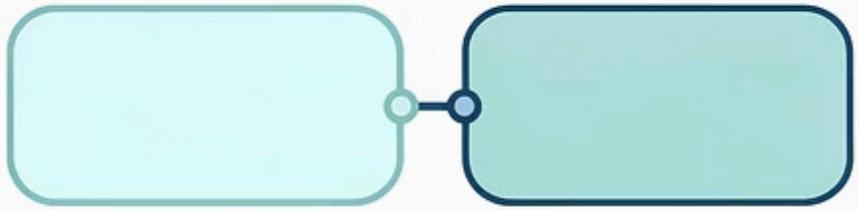
Two entity types. Example: Student enrolls in Course. Most common type in database design.

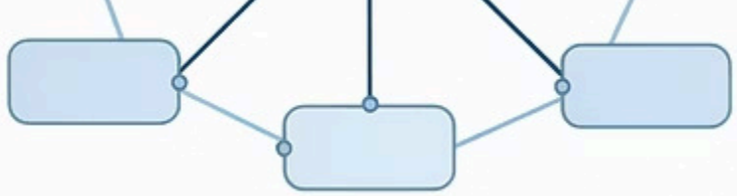
Ternary (Degree 3)

Three entity types. Example: Supplier supplies Part to Project

N-ary (Degree N)

N entity types. Example: Complex relationships involving multiple entities, though these are rare and often decomposed into binary relationships.





Mapping Cardinality

Mapping cardinality defines the number of entity instances that can be associated with another entity instance through a relationship. This constraint is crucial for database design and implementation.

One-to-One (1:1)

Each entity in A relates to at most one entity in B, and vice versa. Example: Person has Passport - one person has exactly one passport.

One-to-Many (1:N)

An entity in A can relate to multiple entities in B, but B relates to only one A. Example: Department has Employees - one department has many employees.

Many-to-Many (M:N)

Entities in A can relate to multiple entities in B, and vice versa. Example: Students enroll in Courses - students take multiple courses, courses have multiple students.

Keys: Uniquely Identifying Records

Keys are attributes or sets of attributes that uniquely identify entity instances and establish relationships between tables. Proper key selection is fundamental to database integrity and performance.

Primary Key

Uniquely identifies each record in a table. Cannot be null. Example: StudentID in Student table

Foreign Key

References primary key of another table, establishing relationships. Example: DepartmentID in Employee table

Candidate Key

Attribute(s) that could serve as primary key. Example: Email, StudentID both uniquely identify students

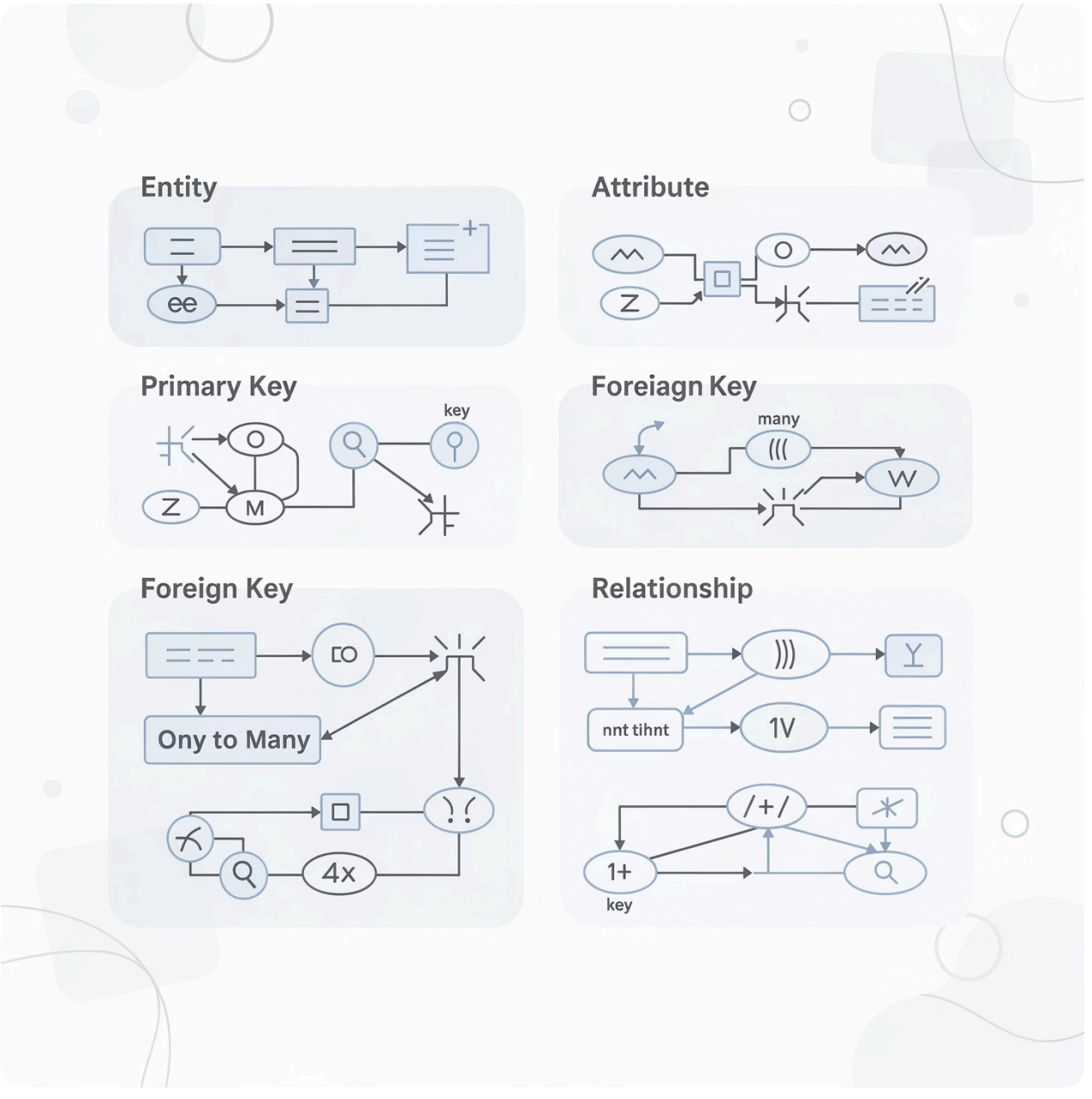
Super Key

Any combination of attributes that uniquely identifies records. Example: {StudentID, Name, Email}

Best Practice: Choose simple, stable attributes as primary keys. Avoid using attributes that might change over time.

Creating ER Diagrams

ER diagrams visually represent database structure using standardized symbols and notations. They serve as blueprints for database implementation and communication tools between stakeholders.



Rectangles

Represent entities (tables)

Ovals

Represent attributes (columns)

Diamonds

Represent relationships between entities

Lines

Connect entities to relationships and attributes

Weak Entity Sets

Weak entities cannot be uniquely identified by their attributes alone and depend on a strong entity (owner entity) for their existence. They require the primary key of their owner entity combined with a partial key to form a complete identifier.

Characteristics

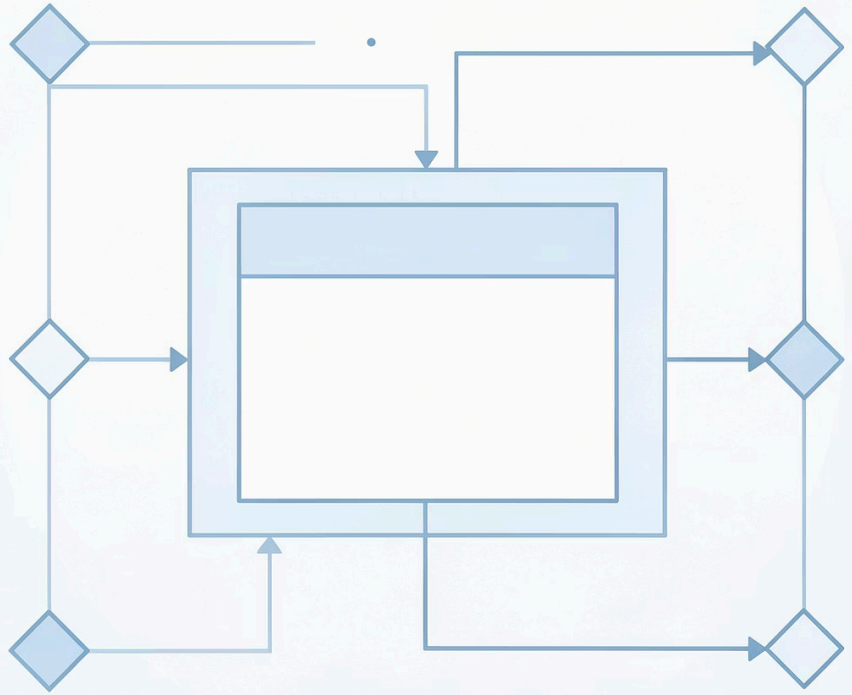
- No primary key of its own
- Depends on strong entity
- Uses partial key (discriminator)
- Represented by double rectangle
- Identifying relationship shown by double diamond

Example: Order & Order Line

Strong Entity: Order (OrderID)

Weak Entity: Order Line (OrderID + LineNumber)

Order Lines cannot exist without an Order, and LineNumber alone doesn't uniquely identify a line item across all orders.



Enhanced ER Model

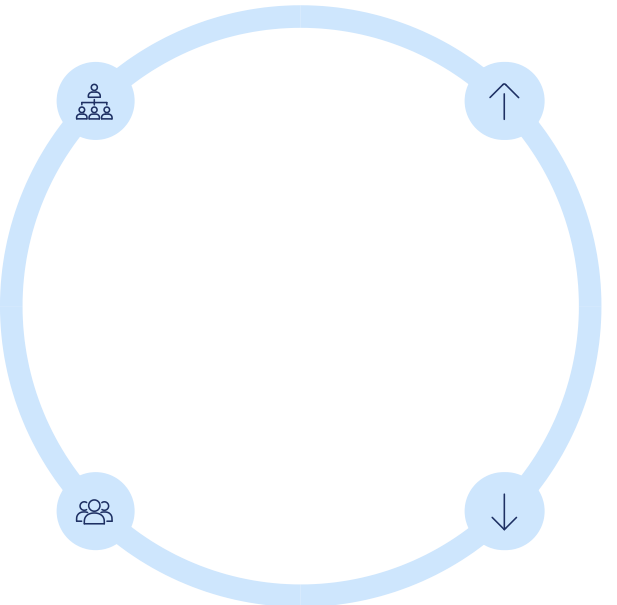
The Enhanced Entity-Relationship (EER) model extends the basic ER model with additional concepts to model more complex scenarios. These extensions provide greater semantic expressiveness for sophisticated database designs.

Subclass & Superclass

Inheritance hierarchies

Generalization

Bottom-up abstraction



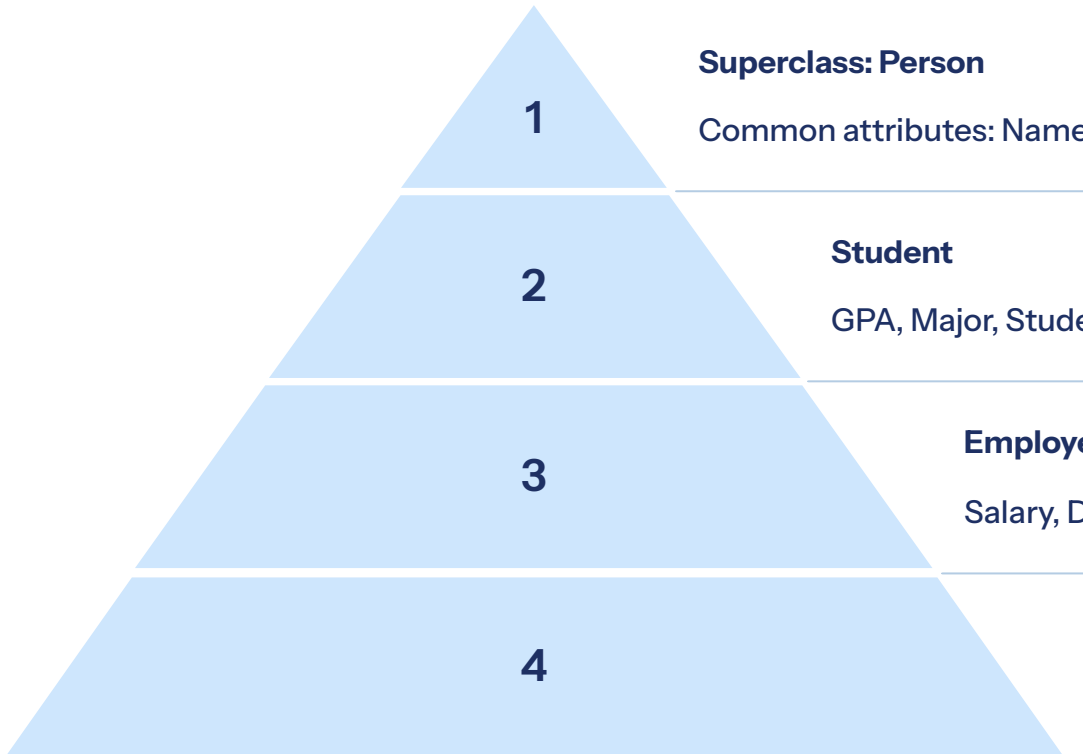
Aggregation

Treating relationships as entities

Specialization

Top-down refinement

Subclass and Superclass



Superclass: Person

Common attributes: Name, DOB, Address

Student

GPA, Major, StudentID

Employee

Salary, Department, EmployeeID

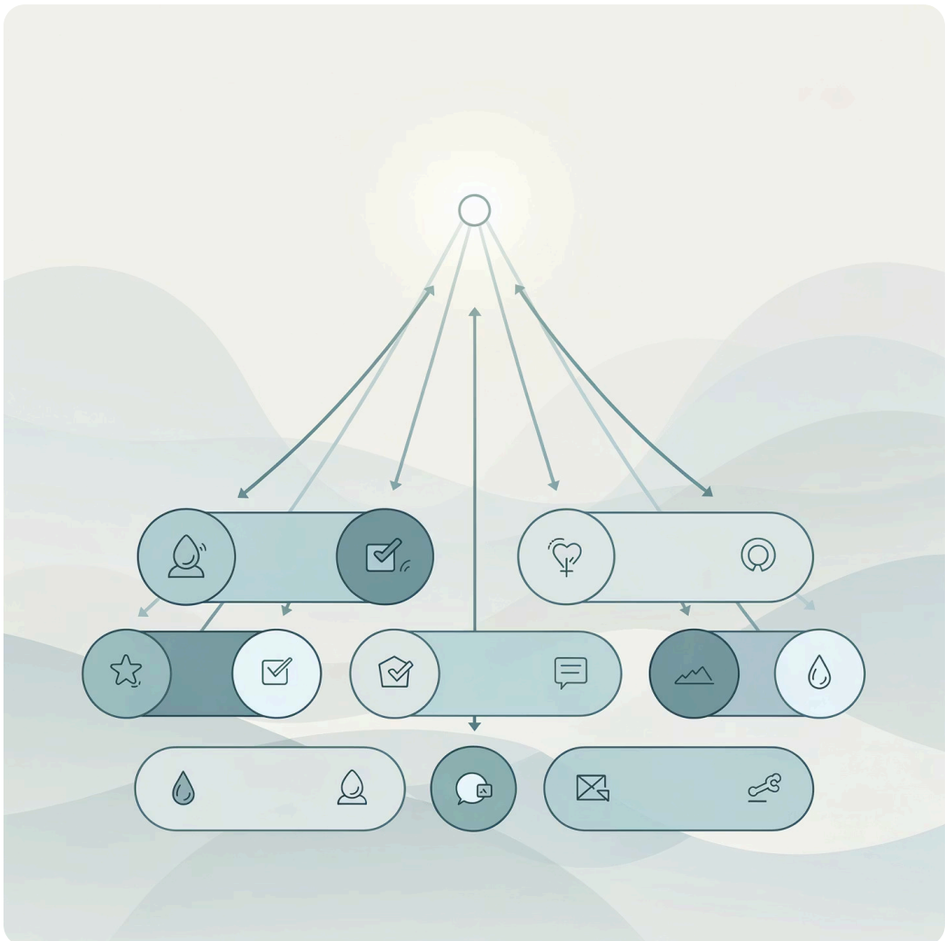
Alumni

Graduation Year, Degree

Subclasses inherit all attributes and relationships from their superclass while adding their own specialized attributes. This models "is-a" relationships where students, employees, and alumni are all types of people.

Generalization vs. Specialization

Generalization

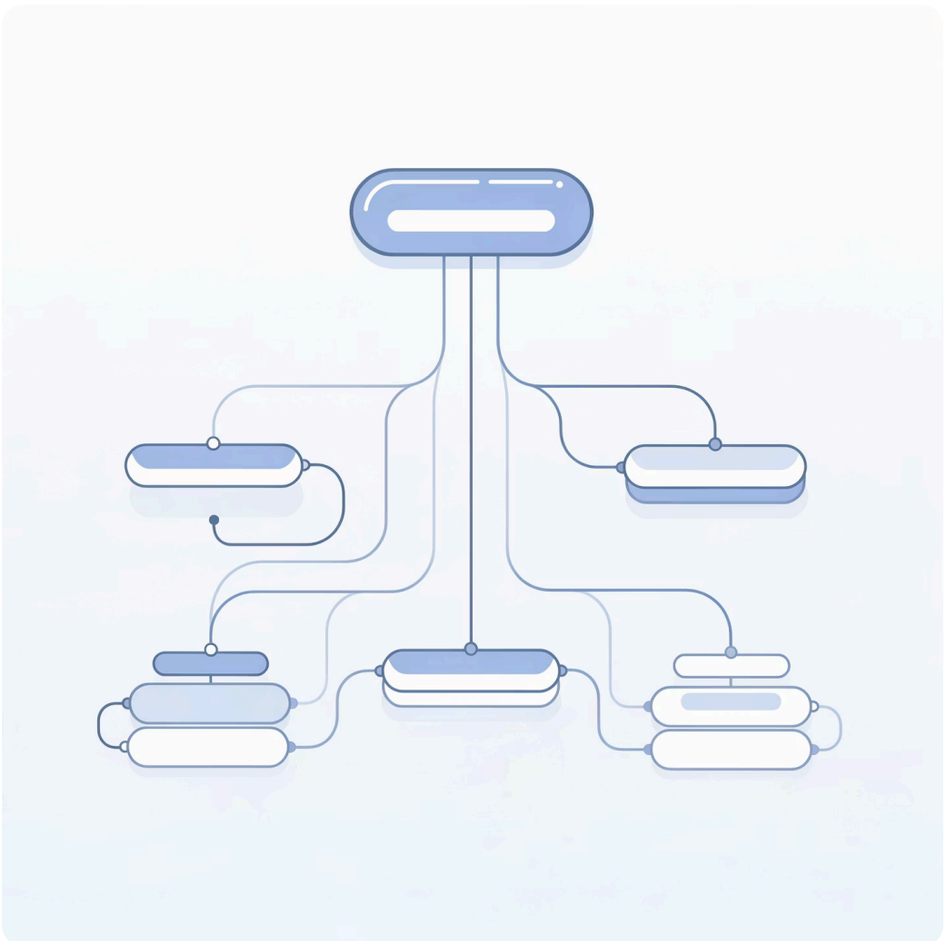


Bottom-up approach that combines multiple entity types with common characteristics into a higher-level superclass.

Process: Identify common attributes among entities → Create superclass → Define inheritance

Example: Car, Truck, Motorcycle → Vehicle

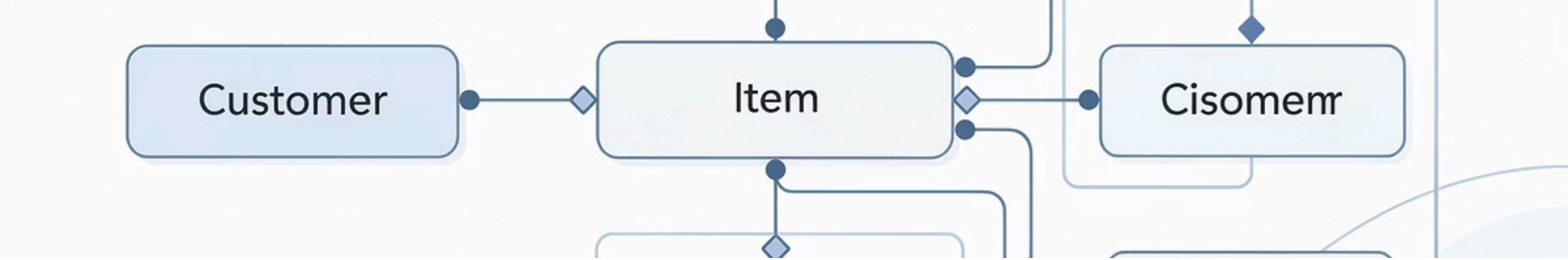
Specialization



Top-down approach that divides a higher-level entity into specialized subclasses based on distinguishing characteristics.

Process: Start with superclass → Identify specialized subgroups → Create subclasses

Example: Account → Savings Account, Checking Account



Aggregation: Relationships as Entities

Aggregation allows treating a relationship as an entity, enabling relationships between relationships. This powerful concept models complex scenarios where relationships themselves participate in other relationships.

> When relationships need to relate to other entities, aggregation provides the solution.

Scenario Example

A project involves employees working on tasks. We need to track which manager supervises each employee-task assignment.

Aggregation Solution

Treat "Employee works on Task" as an aggregate entity, then relate it to Manager through supervision relationship.

Converting ER Diagrams to Database Tables

The final step in database design translates conceptual ER diagrams into physical database schemas. This systematic conversion process creates tables, defines columns, establishes keys, and implements relationships.

Map Strong Entities

Create table for each entity with columns for attributes. Choose primary key.

Map Weak Entities

Include owner's primary key as foreign key. Combine with partial key for primary key.

Map Relationships

1:1 and 1:N use foreign keys. M:N require junction tables with both primary keys.

Map Attributes

Multi-valued attributes become separate tables. Composite attributes are flattened or split.

Implement Constraints

Add NOT NULL, UNIQUE, CHECK, and foreign key constraints to enforce integrity.

UNIT 3

Structured Query Language (SQL)

SQL is the standard language for managing relational databases. This comprehensive unit covers everything from basic data manipulation to advanced queries, functions, and constraints that enforce data integrity.

12

Contact Hours

Most extensive unit

32%

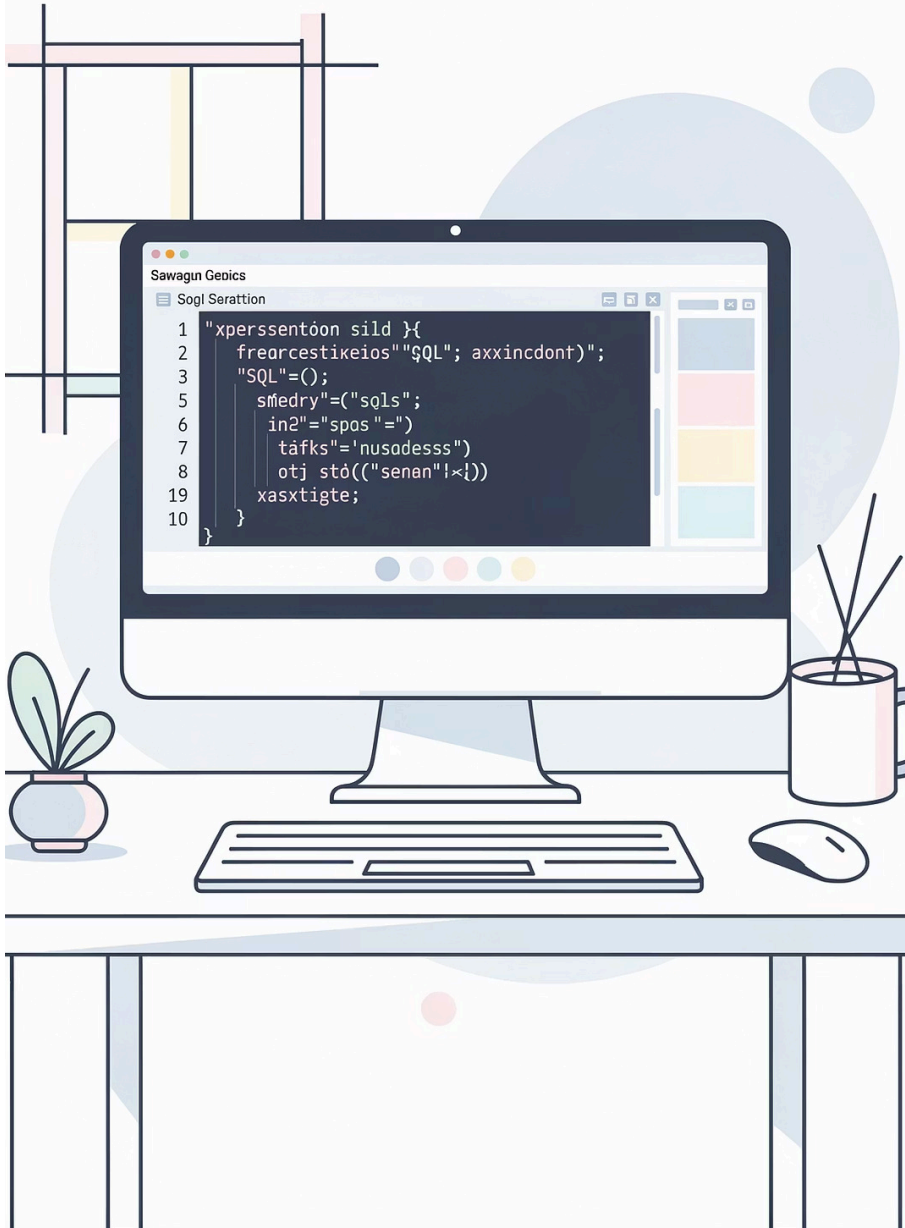
Assessment Weight

Highest priority unit

20+

SQL Topics

Comprehensive coverage



SQL Data Types

Data types define what kind of values a column can store and how much storage space it requires. Choosing appropriate data types is crucial for database performance, storage efficiency, and data integrity.

Numeric Types

- INT, SMALLINT, BIGINT - whole numbers
- DECIMAL, NUMERIC - exact decimals
- FLOAT, REAL - approximate decimals

Character Types

- CHAR(n) - fixed length
- VARCHAR(n) - variable length
- TEXT - long text data

Date & Time Types

- DATE - calendar date
- TIME - time of day
- DATETIME, TIMESTAMP - date and time

Boolean & Binary

- BOOLEAN - true/false
- BLOB - binary large objects
- BIT - bit strings

Data Definition Language (DDL)

DDL commands define and modify database structure. They create, alter, and delete database objects like tables, indexes, and views, establishing the schema that holds your data.

CREATE

Creates new database objects like tables, indexes, or views

```
CREATE TABLE Students (
  StudentID INT PRIMARY KEY,
  Name VARCHAR(50),
  Major VARCHAR(30)
);
```

ALTER

Modifies existing database structures by adding, modifying, or dropping columns

```
ALTER TABLE Students
ADD Email VARCHAR(100);
```

DROP

Permanently deletes database objects and all their data

```
DROP TABLE Students;
```

TRUNCATE

Removes all rows from table but keeps structure intact

```
TRUNCATE TABLE Students;
```

Data Manipulation Language (DML)

DML commands manipulate data within database tables. These are the most frequently used SQL statements, allowing you to insert new records, update existing data, retrieve information, and delete unwanted records.

INSERT

Add new records to tables

SELECT

Retrieve data from tables

UPDATE

Modify existing records

DELETE

Remove records from tables

INSERT Example

```
INSERT INTO Students
(StudentID, Name, Major)
VALUES
(1001, 'John Smith', 'Computer Science');
```

SELECT Example

```
SELECT Name, Major
FROM Students
WHERE Major = 'Computer Science';
```

Privilege Commands: GRANT & REVOKE

Database security relies on controlling who can access and modify data. GRANT and REVOKE commands manage user permissions, implementing the principle of least privilege by providing only necessary access rights.

Proper privilege management protects sensitive data, prevents unauthorized modifications, and maintains audit trails for compliance requirements.

GRANT

Assigns privileges to users or roles

```
GRANT SELECT, INSERT
ON Students
TO 'student_user';
```

REVOKE

Removes previously granted privileges

```
REVOKE INSERT
ON Students
FROM 'student_user';
```

SQL Views: Virtual Tables

Views are virtual tables created by stored queries. They don't store data themselves but present data from one or more underlying tables in a customized format. Views simplify complex queries, enhance security, and provide data abstraction.

Security

Hide sensitive columns, show only authorized data to specific users

Simplification

Present complex joins as simple tables for easier querying

Customization

Provide different perspectives of the same data for different users

```
CREATE VIEW ActiveStudents AS
SELECT StudentID, Name, Major, GPA
FROM Students
WHERE Status = 'Active' AND GPA >= 2.0;
```

SQL Functions Overview

SQL functions perform operations on data and return results. They range from simple calculations to complex string manipulations and date operations, enabling powerful data transformation and analysis within queries.

Character Functions

Manipulate text strings

Numeric Functions

Perform mathematical operations

Miscellaneous

Special purpose operations

Date Functions

Work with dates and times

Group Functions

Aggregate multiple rows

Conversion Functions

Change data types

Character Functions

Character functions manipulate text data, enabling string operations like concatenation, extraction, case conversion, and pattern matching. These functions are essential for data cleaning, formatting, and text analysis.

Function	Purpose	Example
UPPER()	Convert to uppercase	UPPER('hello') → 'HELLO'
LOWER()	Convert to lowercase	LOWER('WORLD') → 'world'
CONCAT()	Combine strings	CONCAT('Hello', ' World')
SUBSTRING()	Extract portion of string	SUBSTRING('Database', 1, 4) → 'Data'
LENGTH()	Get string length	LENGTH('SQL') → 3
TRIM()	Remove spaces	TRIM(' text ') → 'text'

Numeric Functions

Numeric functions perform mathematical calculations on number data types. They handle everything from basic arithmetic to complex calculations, rounding, and absolute values.

ROUND() ROUND(123.456, 2) → 123.46	**CEIL() / FLOOR()** CEIL(4.3) → 5, FLOOR(4.8) → 4
ABS() ABS(-15) → 15	**POWER()** POWER(2, 3) → 8
SQRT() SQRT(16) → 4	**MOD()** MOD(10, 3) → 1

Date Functions

Date functions manipulate temporal data, enabling calculations with dates and times. They extract components, perform date arithmetic, and format date displays—critical for scheduling, reporting, and time-based analysis.

CURRENT_DATE

Returns today's date

```
SELECT CURRENT_DATE;
```

CURRENT_TIME

Returns current time

```
SELECT CURRENT_TIME;
```

DATE_ADD()

Add interval to date

```
DATE_ADD(date, INTERVAL 7 DAY)
```

- **EXTRACT():** Extract year, month, day from date - EXTRACT(YEAR FROM date)
- **DATEDIFF():** Calculate difference between two dates in days
- **DATE_FORMAT():** Format date for display - DATE_FORMAT(date, '%Y-%m-%d')

Conversion Functions

Conversion functions transform data from one type to another, enabling compatibility between different data types in expressions and comparisons. They handle implicit and explicit type conversions safely.

CAST()

SQL standard conversion

```
CAST('123' AS INT)
CAST(price AS DECIMAL(10,2))
```

CONVERT()

Database-specific conversion

```
CONVERT(VARCHAR, date, 101)
CONVERT(INT, '456')
```

TO_DATE()

String to date conversion

```
TO_DATE('2024-01-15', 'YYYY-MM-DD')
```

Best Practice: Always use explicit conversion functions rather than relying on implicit conversion to avoid unexpected results.

Group Functions (Aggregate Functions)

Group functions operate on sets of rows to return a single summary value. They're essential for statistical analysis, reporting, and deriving insights from large datasets.

5

Core Functions

COUNT, SUM, AVG, MAX, MIN

1

Result per Group

Single value from multiple rows

COUNT()

Number of rows

SUM()

Total of values

AVG()

Average value

MAX()

Highest value

MIN()

Lowest value

```
SELECT Major, AVG(GPA) as AvgGPA, COUNT(*) as StudentCount
FROM Students
GROUP BY Major;
```

SQL Operators

Operators perform operations on values and variables in SQL statements. Understanding operator precedence and behavior is crucial for writing correct queries and avoiding logic errors.

Arithmetic Operators

+ (addition), - (subtraction), * (multiplication), / (division), % (modulo)

```
SELECT Price * Quantity AS Total
FROM OrderItems;
```

Comparison Operators

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

```
WHERE Age >= 18 AND GPA > 3.0
```

Logical Operators

AND (both conditions true), OR (either condition true), NOT (negates condition)

```
WHERE (Major = 'CS' OR Major = 'IT')
AND NOT (Status = 'Inactive')
```

GROUP BY, HAVING, and ORDER BY

These clauses organize and filter query results, enabling sophisticated data analysis and presentation. They work together to group data, filter groups, and sort output.

01

GROUP BY

Divides rows into groups based on column values for aggregate functions

02

HAVING

Filters groups based on aggregate conditions (WHERE filters rows, HAVING filters groups)

03

ORDER BY

Sorts final result set in ascending (ASC) or descending (DESC) order

```
SELECT Department, AVG(Salary) as AvgSalary
FROM Employees
GROUP BY Department
HAVING AVG(Salary) > 50000
ORDER BY AvgSalary DESC;
```

Course Journey Complete

You've covered the comprehensive Database Management System curriculum from foundational concepts to advanced SQL operations. This knowledge forms the backbone of modern information systems and prepares you for real-world database challenges.

Strong Foundation

Database concepts, architecture, and administration principles

Design Skills

ER modeling, normalization, and schema conversion techniques

SQL Mastery

Comprehensive command of database querying and manipulation

Data Integrity

Transaction management and constraint implementation

Continue practicing with real-world scenarios, explore advanced topics like stored procedures and triggers, and apply these concepts in your projects. The database skills you've acquired are in high demand across the IT industry.