

UNIT 2

Database Design, Architecture and Model

(6 Hours) — Database Management System (CACS255) · BCA II Year / IV Semester

⚠ Important Note — Please Read

These notes were prepared quickly from the course syllabus and standard textbook knowledge. **They may contain errors, typing mistakes or incomplete points.** Please take care: cross-check definitions, formulas, SQL syntax and examples with your class notes, the prescribed textbook and your teacher before relying on them for exams or practicals.

2.1 Overview of the Database Design Process

Designing a database is done step by step. A good design saves storage, avoids errors and makes queries fast.

1. **Requirement analysis** — collect what data users need and what they want to do with it.
2. **Conceptual design** — draw a high-level model, usually an **E-R diagram**, independent of any DBMS.
3. **Logical design** — convert the E-R diagram into relational tables and apply normalization.
4. **Physical design** — choose file organization, indexes and storage details.
5. **Implementation and maintenance** — create tables using SQL, load data, test, tune and maintain.

View of Data

A major purpose of a DBMS is to give users an **abstract view** of data, hiding how it is stored. There are three levels of abstraction:

Level	Meaning	Who sees it
Physical level	How data is actually stored (files, blocks, indexes).	DBA, system programmers
Logical level	What data is stored and the relationships among them.	DBA, designers, programmers
View level	Only the part of the database relevant to a user.	End users

- **Schema:** the overall design (structure) of the database; it rarely changes.
- **Instance:** the collection of data actually stored at a particular moment; it changes often.

2.2 Structure of Database Management System

A DBMS has two main parts — the **Query Processor** and the **Storage Manager** — with disk storage below them.

Part	Components and role
Query Processor	DDL interpreter (records definitions in the data dictionary), DML compiler (converts DML to low-level instructions and optimizes), Query evaluation engine (executes the plan).
Storage Manager	Authorization & integrity manager , Transaction manager (ensures consistency during failures and concurrency), File manager (allocates disk space), Buffer manager (moves data between disk and main memory).
Disk storage	Data files , data dictionary (metadata), indices and statistical data .

2.3 Three-Level Database Architecture and Data Independence

The **ANSI/SPARC three-level (three-schema) architecture** separates user views from physical storage:

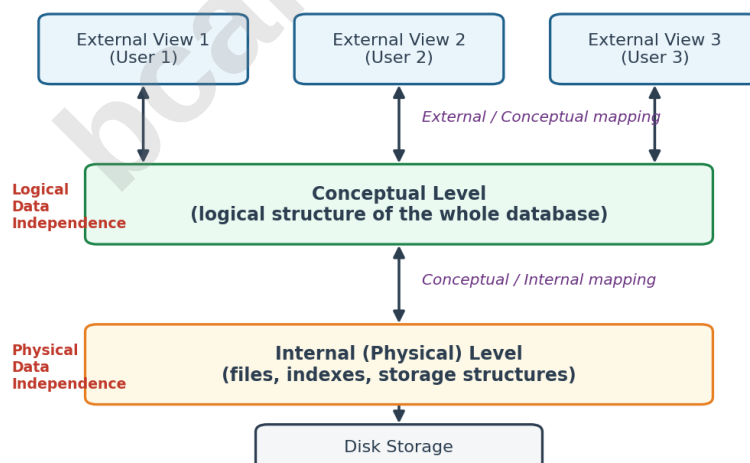


Figure 2.1 — Three-level architecture of a DBMS

- **External level:** many user views; each view shows only the data required by a group of users.
- **Conceptual level:** one community view of the whole database — entities, attributes, relationships and constraints.
- **Internal level:** physical storage — file structures, indexes, record placement, compression.

Data Independence

Data independence is the ability to change the schema at one level without changing the schema at the next higher level.

Type	Definition	Example
Physical data independence	Change internal schema without changing the conceptual schema.	Adding an index or moving a file to a new disk.
Logical data independence	Change conceptual schema without changing external views/programs.	Adding a new column to a table.

Note: Logical data independence is harder to achieve than physical data independence.

2.4 Database Languages

Language	Full form	Purpose and commands
DDL	Data Definition Language	Defines structure: CREATE, ALTER, DROP, TRUNCATE, RENAME
DML	Data Manipulation Language	Access and change data: SELECT, INSERT, UPDATE, DELETE
DCL	Data Control Language	Permissions: GRANT, REVOKE
TCL	Transaction Control Language	COMMIT, ROLLBACK, SAVEPOINT
QBE	Query By Example	Graphical/tabular way of querying by filling a skeleton table

- **Procedural DML** — the user tells *what* data is needed and *how* to get it (e.g. relational algebra).
- **Non-procedural (declarative) DML** — the user tells only *what* is needed (e.g. SQL).

Query By Example (QBE)

QBE was developed at IBM. The user writes an example of the answer in a skeleton table. **P.** means *print/display*, and an underlined value (or a variable such as _x) is an example element.

Example 2.1 — QBE query

Query: Display the names of all students of the CS department.

Skeleton table **Student** with columns (SID, Name, Dept): put **P.** under *Name* and the constant **CS** under *Dept*.

This is equivalent to SQL: *SELECT Name FROM Student WHERE Dept = 'CS';*

2.5 Data Models

A **data model** is a collection of concepts for describing data, relationships, semantics and constraints.

Model	Structure	Key features
Hierarchical	Tree — one parent, many children	Fast for 1:N; cannot represent M:N easily; e.g. IBM IMS.
Network	Graph — a record can have many parents	Handles M:N using set types/pointers; complex navigation; e.g. IDS, CODASYL.
Relational	Tables (relations) of rows and columns	Simple, based on set theory; uses SQL; e.g. MySQL, Oracle, SQL Server.
E-R Model	Entities, attributes, relationships	High-level conceptual design; drawn as E-R diagram.
Object-Based	Objects with attributes and methods; classes, inheritance	Good for complex data (multimedia, CAD); e.g. object-oriented and object-relational databases.

2.6 Entity-Relationship (E-R) Diagram

Basic Concepts

- **Entity:** a real-world thing that can be identified (Student, Course, Teacher).
- **Entity set:** collection of entities of the same type (all students).
- **Attribute:** a property of an entity (Name, Age).
- **Relationship:** an association among entities (Student *enrolls in* Course).

Attribute type	Meaning	Example
Simple / Atomic	Cannot be divided further	Age, Gender
Composite	Can be divided into sub-parts	Name → First, Middle, Last
Single-valued	Only one value per entity	Date of birth

Attribute type	Meaning	Example
Multi-valued	Many values per entity (double ellipse)	Phone numbers
Derived	Calculated from other attributes (dashed ellipse)	Age from DOB
Key attribute	Uniquely identifies an entity (underlined)	StudentID

Relationship Concepts

- **Degree:** number of entity sets in a relationship — unary (1), binary (2), ternary (3).
- **Mapping cardinality:** **1:1** (Person–Passport), **1:N** (Department–Employee), **N:1**, **M:N** (Student–Course).
- **Participation:** *total* (every entity must take part — double line) or *partial* (single line).

E-R Diagram Symbols

Symbol	Represents
Rectangle	Entity set
Double rectangle	Weak entity set
Ellipse	Attribute
Double ellipse	Multi-valued attribute
Dashed ellipse	Derived attribute
Diamond	Relationship set
Double diamond	Identifying relationship of a weak entity
Line	Links attributes to entity sets and entity sets to relationships

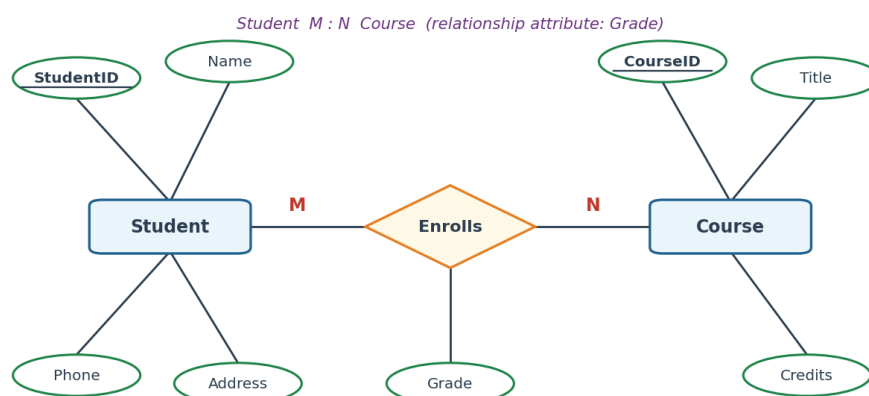


Figure 2.2 — E-R diagram of Student and Course with M:N relationship “Enrolls”

Strong and Weak Entity Sets

- **Strong entity set:** has a primary key of its own (Student).
- **Weak entity set:** has no primary key of its own; it depends on an **owner (strong) entity**. It is identified by the owner's key plus its own **partial key (discriminator)**. Example: *Dependent of an Employee, or Room of a Building.*

Generalization, Specialization and Aggregation

- **Specialization (top-down):** a higher-level entity is divided into sub-groups. *Employee* → *Teacher, Clerk.*
- **Generalization (bottom-up):** common features of several entities are combined into a higher-level entity. *Car, Bus, Truck* → *Vehicle.* Drawn with a triangle labelled **ISA**.
- **Aggregation:** a relationship between entities is treated as a higher-level entity so it can take part in another relationship. Example: *Employee works-on Project* is aggregated, then related to *Machinery* through “uses”.

Note: Constraints on generalization: disjoint vs. overlapping, and total vs. partial completeness.

Converting E-R Diagrams to Tables

E-R element	Conversion rule
Strong entity	Becomes a table; each simple attribute becomes a column; key attribute becomes primary key.
Composite attribute	Include only its component parts as separate columns.
Multi-valued attribute	Create a new table with the owner's key + the attribute; both together form the primary key.
Weak entity	Table with its partial key + owner's primary key (as foreign key); together they form the primary key.
1:1 relationship	Put the primary key of one side as a foreign key on the other side.
1:N relationship	Put the primary key of the “1” side as a foreign key in the “N” side table.
M:N relationship	Create a new table containing the primary keys of both entities (+ relationship attributes).
Generalization	Table for the super-class and tables for sub-classes (with the super-class key), or one combined table.

Example 2.2 — Convert the E-R diagram of Figure 2.2 into tables

Entities: Student(StudentID, Name, Address, Phone*), Course(CourseID, Title, Credits);
relationship Enrolls (M:N) with attribute Grade.

Phone is *multi-valued*, so it needs a separate table.

Solution

Student(StudentID, Name, Address)

StudentPhone(StudentID, Phone) — StudentID is a foreign key to Student

Course(CourseID, Title, Credits)

Enrolls(StudentID, CourseID, Grade) — both are foreign keys; together they form the primary key

Key Takeaways — Unit 2

- ✓ Design steps: requirements → conceptual (E-R) → logical (tables) → physical → implementation.
- ✓ Three levels of abstraction: physical, logical and view; three-schema architecture: internal, conceptual, external.
- ✓ Physical independence = change storage without touching the conceptual schema; logical independence = change the conceptual schema without touching user views.
- ✓ DDL defines structure, DML manipulates data, DCL controls permission, TCL manages transactions.
- ✓ Data models: hierarchical, network, relational, E-R and object-based.
- ✓ E-R terms: entity, attribute, relationship, cardinality, weak entity, generalization, aggregation.
- ✓ M:N relationships and multi-valued attributes always need a separate table.

— End of Unit 2 —

Reminder: these notes may contain errors — verify with your textbook and teacher.