

DBMS

Database Management Systems · Job-Ready

ER models to normalization, SQL to window functions, transactions to B+ trees. Simple language, worked examples, and 120+ interview questions with answers.



interview.sql

```
SELECT name, DENSE_RANK() OVER (ORDER BY skill DESC) AS rank
FROM   candidates
WHERE  topic = 'DBMS' AND prepared = TRUE
ORDER BY offer_ctc DESC;  -- 1 row: you
```

Normalization

ACID

Joins

Indexes

B+ Tree

Transactions

Isolation Levels

Window Functions

Deadlocks

NoSQL

Keys

Query Plans

WRITTEN BY

Shreyash Tripathi

JOB-READY SERIES • COMPLETE EDITION



Database Management Systems

Job-Ready Guide for developers preparing for their next job



WRITTEN BY

Shreyash Tripathi

Complete Edition, 2026

© Shreyash Tripathi. All rights reserved.

Preface

You already have one job. You write code, you push to production, and sometimes things break in ways nobody explained in college. Now you want the next job, and the interviews ask the same "basic" Database Management Systems questions again, except now the interviewer also expects you to connect them to real work. This book is written for exactly that moment.

It follows the style of a school textbook on purpose: numbered chapters, simple language, worked examples step by step, a check with answers after every topic, and full exercises with an answer key at the end of every chapter. Every chapter also has an **Interview Corner** with questions interviewers really ask, and an **At Work** box that shows where the concept appears in a real job. There is humour too, because a joke is easier to remember than a definition.

Read one chapter a day, solve the questions before you look at the answers, and revise the Interview Corners the night before your interview. You will walk in prepared.

Shreyash Tripathi

Features of Every Chapter

DO YOU KNOW?

Real facts and history.

FUNNY SIDE

A joke that makes the idea stick.

KEY POINT

The line to remember.

INTERVIEW CORNER

Questions interviewers really ask, with model answers.

AT WORK

How the concept shows up in a real job.

COMMON MISTAKE

Traps that cost marks and cause bugs.

CHECK YOUR PROGRESS

Short questions with answers after every topic.

Each chapter ends with **What Have We Learnt?**, **Key Terms**, **Exercises** and a complete **Answer Key**. The last chapter is a full **Interview Question Bank**.

Contents

PART I: FOUNDATIONS

1	Introduction to DBMS	6
2	Data Modelling and the ER Model	31
3	The Relational Model, Keys and Constraints	55
4	Relational Algebra and Relational Calculus	75

PART II: SQL

5	SQL I: Fundamentals	102
6	SQL II: Joins, Subqueries and Advanced Queries	135
7	SQL III: Interview Problems, Procedures and Triggers	166

PART III: DESIGN AND THEORY

8	Functional Dependencies and Normalization	193
9	Transactions and Serializability	215
10	Concurrency Control	237
11	Recovery and Durability	260

PART IV: INTERNALS AND MODERN DATABASES

12	Indexing, Hashing and File Organization	283
13	Query Processing and Optimization	303
14	NoSQL and Distributed Databases	326
15	Database Security, Administration and Best Practices	349

PART V: INTERVIEW PREPARATION

16	DBMS and SQL Interview Question Bank	375
-----------	--	-----

APPENDICES

A	SQL Cheat Sheet	404
B	Normalization and Keys Quick Reference	410
C	Glossary	414
D	Sample Question Paper with Answers	418

PART I

Foundations

What a DBMS is, how to model data, keys and relational algebra.

CHAPTER 1

Introduction to DBMS

*"A spreadsheet is a database that has not yet had its first production incident."***After studying this chapter, you will be able to**

- Explain the difference between data and information, and between a file system and a DBMS.
- List the problems of file-based storage (redundancy, inconsistency, isolation, integrity, atomicity, concurrency, security) and show how a DBMS solves each one.
- Describe the three-schema architecture and the two kinds of data independence.
- Distinguish schema from instance, and 1-tier, 2-tier and 3-tier architectures.
- Name the main components of a DBMS (query processor, storage manager, buffer manager) and what each does.
- Classify SQL commands into DDL, DML, DCL and TCL, and describe the role of a DBA.
- Compare relational and NoSQL databases, and OLTP and OLAP workloads, and name popular DBMS products.

Think of a busy chai stall near a railway station. In the first month, the owner, Raju bhai, keeps credit records in a small notebook: "Sharma ji, 3 chai, 1 samosa, pay on Friday." It works. Then he opens a second stall at the bus stand, and his nephew keeps a second notebook there. Sharma ji drinks chai at both places. Now his dues are written in two notebooks, one of them is wet with chai, and nobody knows the true total. On Friday there is an argument.

This little story is the whole reason database management systems exist. Once data is shared by many people, in many places, at the same time, and must survive spills, crashes and mistakes, a notebook (or a folder of CSV files) stops being enough. In this chapter we will see exactly what goes wrong without a DBMS, how a DBMS fixes it, and the vocabulary every interviewer expects you to use without thinking: schema, instance, data independence, DDL, DML, OLTP, OLAP and more.

1.1 Data, Information and Databases

Data is raw facts with no context: 42 , "Pune" , 2026-03-14 . **Information** is data that has been processed, organised and given meaning so that someone can make a decision: "Order 42 was shipped to Pune on 14 March 2026, two days late." Data is the flour; information is the roti.

A **database** is an organised collection of related data, stored so that it can be easily accessed, managed and updated. A **Database Management System (DBMS)** is the software that lets users define, create, query, update and protect that database. MySQL, PostgreSQL, Oracle and MongoDB are DBMS software. The data of your company's orders table is the database. Together, the DBMS plus the database plus the applications using them is often called a **database system**.

Term	Meaning	Example
Data	Raw facts	98, 87, 45
Information	Processed, meaningful data	"Average marks of the class is 76.7"
Database	Organised collection of related data	The <code>students</code> and <code>marks</code> tables
DBMS	Software that manages the database	PostgreSQL, MySQL, Oracle
Database system	DBMS + database + applications + users	College ERP running on PostgreSQL

KEY POINT

A database is the data. A DBMS is the software that manages it. Interviewers like to check that you do not mix the two. "MySQL is a database" is casual speech; "MySQL is a DBMS" is the precise answer.

DO YOU KNOW?

The relational model, which almost every SQL database is based on, was proposed by Edgar F. Codd of IBM in his 1970 paper "A Relational Model of Data for Large Shared Data Banks". Codd later received the Turing Award (1981) for this work.

CHECK YOUR PROGRESS 1.1

1. Is "Rahul, 21, Delhi" data or information?

Answer: It is data. It becomes information only when it is given context and meaning, for example "Rahul, aged 21, is the youngest customer from Delhi this month."

2. What is the difference between a database and a DBMS?

Answer: A database is the organised collection of data itself. A DBMS is the software (like PostgreSQL) used to create, query, update and protect that data.

1.2 File System versus DBMS

Before DBMS software became common, programs stored data in plain files. A payroll program had its own file, the HR program had another, and each program contained its own code to read and write its own file format. This is called a **file processing system**. Many small projects still start this way today, with CSV files, Excel sheets or JSON files on disk. It feels simple, until it is not.

1.2.1 Problems with the file-based approach

Let us use a college example. The Accounts office keeps `fees.csv`, and the Exam office keeps `exams.csv`. Both files store the student's name, phone and address.

1. **Data redundancy:** The same data is stored in many places. Student address is in both files. This wastes space, but the bigger cost is the next problem.
2. **Data inconsistency:** Copies disagree. Priya changes her phone number at the Accounts office. The Exam office still has the old number, so her hall ticket SMS goes nowhere. Redundancy causes inconsistency.
3. **Difficulty in accessing data:** To answer a new question ("students from Nagpur with pending fees") someone must write a new program. There is no general query language.
4. **Data isolation:** Data is scattered across files in different formats (CSV here, fixed-width text there, Excel somewhere else). Combining them is painful.
5. **Integrity problems:** Rules like "fees paid cannot be negative" or "marks are between 0 and 100" are buried inside application code. A new program can forget the rule and write bad data.
6. **Atomicity problems:** A transfer of Rs 500 from wallet A to wallet B needs two writes. If the program crashes after debiting A but before crediting B, Rs 500 vanishes. Files cannot guarantee "all or nothing".
7. **Concurrent access anomalies:** Two clerks open the same file, both read balance 1000, one adds 200 and one subtracts 300, both save. The final value is either 1200 or 700, never the correct 900. This is the "lost update" problem.
8. **Security problems:** With files, it is hard to say "the exam clerk may see marks but not fee details". Usually anyone with file access sees everything.

FUNNY SIDE

Every company has a file named `final_data_v3_FINAL_use_this_one(2).xlsx`. That file is a working demonstration of redundancy, inconsistency and concurrency problems, all at once, sitting on a shared drive.

1.2.2 How a DBMS solves these problems

Problem	File system	How a DBMS handles it
Redundancy and inconsistency	Each program keeps its own copy	One shared, centrally designed database; normalisation removes duplicate storage
Difficult access	New program per question	Declarative query language (SQL)
Isolation	Many formats	One data model, one catalogue of all tables
Integrity	Rules hidden in code	Constraints (PRIMARY KEY, FOREIGN KEY, CHECK, NOT NULL) enforced by the DBMS

Problem	File system	How a DBMS handles it
Atomicity	Partial writes on crash	Transactions with COMMIT/ROLLBACK, write-ahead logging and recovery
Concurrency	Lost updates	Locking or MVCC, isolation levels
Security	All or nothing file access	Users, roles, GRANT/REVOKE, row-level security, encryption
Backup and recovery	Manual copying	Built-in backup, point-in-time recovery, replication

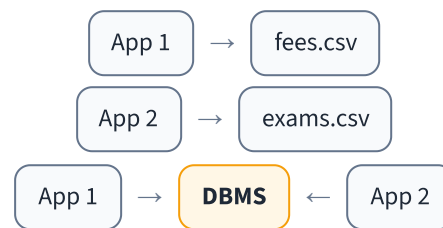


Figure 1.1: Top two rows: each program owns its file. Bottom row: all programs share one DBMS-managed database.

1.2.3 Advantages of a DBMS

- Controlled redundancy and better consistency.
- Data sharing among many users and applications at the same time.
- Enforcement of integrity constraints in one place.
- Transactions: atomicity, consistency, isolation, durability (ACID).
- Security and access control.
- Backup, recovery and replication.
- Data independence: programs do not break when storage details change.
- Powerful querying and reporting with SQL, plus query optimisation done for you.
- Standards: a common language and tools, so a new developer is productive on day one.

1.2.4 Disadvantages of a DBMS

- **Cost:** licences (for commercial products), hardware, and skilled people (DBAs).
- **Complexity:** installation, tuning, upgrades and backups need knowledge.
- **Performance overhead:** logging, locking and constraint checks cost time. For a tiny single-user tool, a flat file can be faster.
- **Single point of failure:** if the central database goes down, every app that uses it goes down, unless you add replication.
- **Size:** the software itself needs memory and disk.

When is a file enough? A config file, a log file, a one-time data export, or a script that runs on your laptop. When many users write concurrently, money is involved, or data must not be lost, use a DBMS.

 DO YOU KNOW?

SQLite is an embedded DBMS: it runs inside your application process with no separate server. Its website states it is very likely the most widely deployed database engine in the world, because it ships inside every Android and iOS phone, most web browsers and many other products.

 AT WORK

A startup kept user wallet balances in a Redis key and updated them with "read, add, write back" from two different services. During a sale, two refunds for the same user landed at the same time and one was lost: a classic concurrent access anomaly. The fix was to move balances into PostgreSQL and update them inside a transaction with `UPDATE wallets SET balance = balance + 200 WHERE user_id = 7`, which the DBMS executes atomically with a row lock. Knowing the textbook list of file-system problems helps you name a bug quickly in a postmortem.

 INTERVIEW CORNER

Q: Why use a DBMS instead of storing data in files?

A: Files suffer from redundancy, inconsistency, difficult access, isolation, integrity, atomicity, concurrency and security problems. A DBMS gives a shared, consistent database with a query language, constraints, ACID transactions, concurrency control, access control and recovery.

Q: Give one disadvantage of a DBMS.

A: Overhead and cost: logging, locking and constraint checks add latency, and the system needs skilled administration. For a tiny single-user tool, plain files may be simpler.

 CHECK YOUR PROGRESS 1.2

1. How is data redundancy related to data inconsistency?

Answer: Redundancy means the same fact is stored in several places. When one copy is updated and others are not, the copies disagree, which is inconsistency. So redundancy creates the opportunity for inconsistency.

2. Which file-system problem is shown by "debit happened but credit did not, after a crash"?

Answer: The atomicity problem. A DBMS solves it with transactions: either both writes happen, or neither does.

3. Name two features of a DBMS that solve concurrency problems.

Answer: Locking (for example row-level locks) and MVCC (multi-version concurrency control), controlled through transaction isolation levels.

1.3 The Three-Schema Architecture

A database is used by many people who want different views of the same data. The fee clerk wants student names and dues. The exam office wants roll numbers and marks. The DBA cares about which disk file and which index stores each row. To keep these concerns apart, the ANSI/SPARC

committee proposed (in the 1970s) the **three-schema architecture**, also called the three-level architecture.

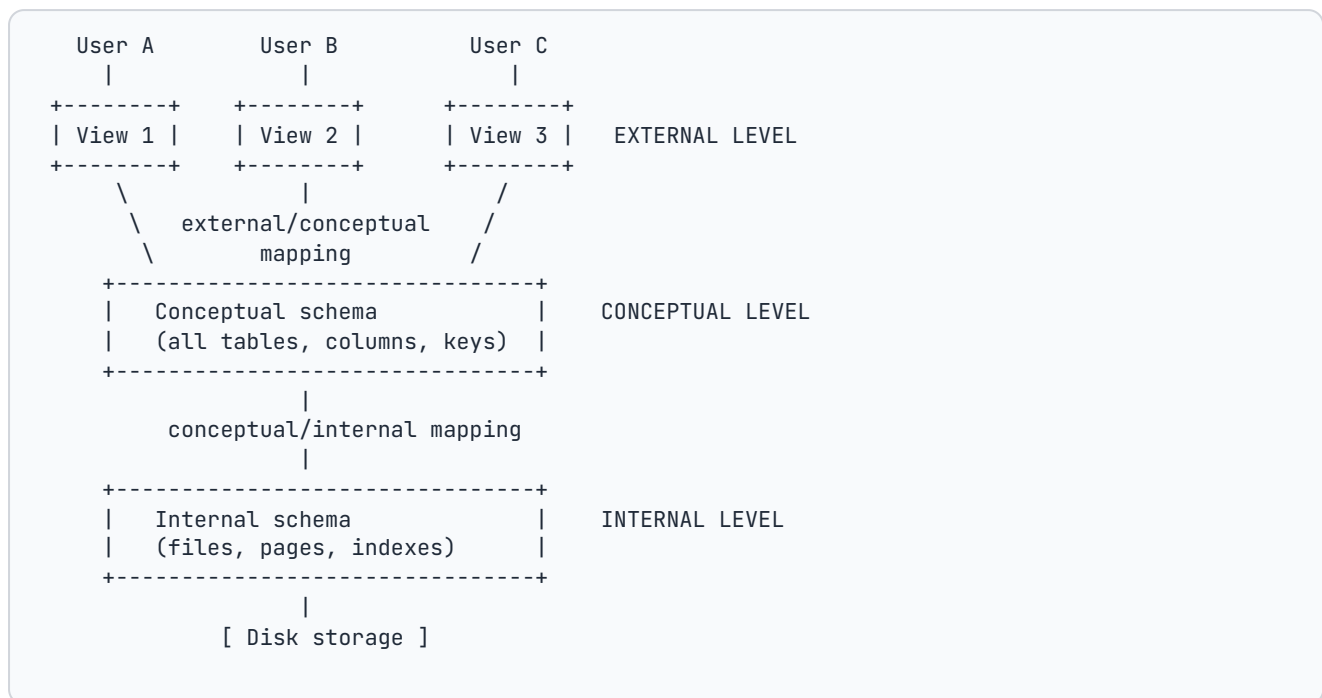


Figure 1.2: The three-schema architecture

1.3.1 Internal (physical) level

The lowest level describes **how** data is physically stored: file organisation, page size, record layout, indexes (B+ tree or hash), compression and where the data sits on disk. For example, "the `students` table is stored as a heap file with 8 KB pages, and there is a B+ tree index on `roll_no`." Only the DBMS and the DBA care about this level.

1.3.2 Conceptual (logical) level

The middle level describes **what** data is stored for the whole organisation and how it is related: all the tables, columns, data types, keys, relationships and constraints. It hides physical details. For example, "`students(roll_no PK, name, dept_id FK → departments)`". Developers and DBAs work at this level. There is exactly one conceptual schema per database.

1.3.3 External (view) level

The top level describes the part of the database that a particular user group sees. There can be many external schemas. In SQL they are usually implemented as **views** and permissions. For example, a view `fee_defaulters(name, phone, due_amount)` for the Accounts office, which hides marks completely.

```
-- An external schema for the Accounts office
CREATE VIEW fee_defaulters AS
SELECT s.name, s.phone, f.due_amount
FROM students s JOIN fees f ON f.roll_no = s.roll_no
WHERE f.due_amount > 0;

GRANT SELECT ON fee_defaulters TO accounts_clerk;
```

1.3.4 Mappings

The DBMS keeps two mappings: **external/conceptual** (how each view is computed from the tables) and **conceptual/internal** (how each table is laid out in storage). When a user queries a view, the DBMS translates the request down through both mappings to actual disk reads.

FUNNY SIDE

A wedding buffet is a three-schema architecture. Guests see the external level: a beautiful counter with paneer and gulab jamun. The caterer's menu card is the conceptual level: every dish, every ingredient. The kitchen behind the tent, with gas cylinders and huge vessels, is the internal level. Guests are much happier not seeing the internal level.

✓ CHECK YOUR PROGRESS 1.3

1. At which level would you find information about a B+ tree index on a column?

Answer: The internal (physical) level, because it describes how data is stored and accessed on disk.

2. How many conceptual schemas and how many external schemas can a database have?

Answer: Exactly one conceptual schema, but many external schemas (one per user group or application view).

3. How is an external schema usually implemented in SQL databases?

Answer: Through views combined with privileges (GRANT), so each user group sees only the columns and rows meant for it.

1.4 Data Independence

Data independence is the ability to change the schema at one level without having to change the schema at the next higher level. It is the main payoff of the three-schema architecture. Without it, every storage tweak would mean rewriting application code.

1.4.1 Physical data independence

The ability to change the **internal** schema without changing the conceptual schema or applications. Examples: adding an index, moving a table to a faster SSD, changing the file organisation, partitioning a table, or compressing pages. Your `SELECT * FROM students WHERE roll_no = 7` still works; it just runs faster. Physical data independence is fairly easy to achieve, and all modern relational DBMSs provide it well.

1.4.2 Logical data independence

The ability to change the **conceptual** schema without changing external schemas or applications. Examples: adding a new column, adding a new table, or splitting one table into two. If the `students` table is split into `students_basic` and `students_contact`, you can create a view called `students` that joins them back, so old queries keep working. Logical independence is harder to achieve, because applications are more directly tied to the logical structure (for example, code doing `SELECT *` breaks when a column is added or removed).

Aspect	Physical data independence	Logical data independence
What changes	Internal schema	Conceptual schema
What stays unchanged	Conceptual schema and apps	External schemas (views) and apps
Example change	Add index, change storage engine, partition	Add column, split table, add table
Difficulty	Easier	Harder
Mapping involved	Conceptual/internal	External/conceptual

⚠ COMMON MISTAKE

Writing `SELECT *` and `INSERT INTO t VALUES (...)` without column names in application code. These destroy logical data independence: the day someone adds a column, the insert fails or the code maps values to wrong fields. Always name your columns.

📦 AT WORK

A team adds an index on `orders(customer_id)` in production using `CREATE INDEX CONCURRENTLY` in PostgreSQL. No code is deployed, yet the "My Orders" page goes from 2 seconds to 20 ms. That is physical data independence in action: the storage changed, the queries did not.

✅ CHECK YOUR PROGRESS 1.4

1. Adding a new index to a table is an example of which data independence?

Answer: Physical data independence, because only the internal (storage) schema changes and applications are unaffected.

2. Why is logical data independence harder to achieve than physical?

Answer: Applications directly depend on table and column names and structure. Changing the logical structure can break queries unless views are used to hide the change, while physical changes are fully hidden by the DBMS.

1.5 Schema and Instance

A **schema** is the overall design or structure of the database: table names, columns, types, constraints. It is defined once and changes rarely. An **instance** (also called a **database state** or

snapshot) is the actual data stored in the database at a particular moment. It changes every time a row is inserted, updated or deleted.

Analogy: the IRCTC train chart format (coach, berth, name, age, status) is the schema. The actual chart printed for train 12951 on a given day is an instance. Tomorrow's chart is a different instance with the same schema.

```
-- Schema (structure)
CREATE TABLE students (
  roll_no INT PRIMARY KEY,
  name    VARCHAR(50) NOT NULL,
  city    VARCHAR(30)
);

-- Instance at 10:00 AM (the data)
-- roll_no | name  | city
-- 1       | Asha  | Pune
-- 2       | Kabir | Indore
```

Related terms: the schema is sometimes called the **intension** and the instance the **extension** of the database. The schema is stored in the DBMS **data dictionary** (system catalogue), for example `information_schema` in MySQL and PostgreSQL.

ACTIVITY 1.1

Open any PostgreSQL or MySQL database (a free online playground is fine) and run `SELECT table_name, column_name, data_type FROM information_schema.columns WHERE table_schema = 'public';` (use your database name instead of 'public' in MySQL). You are reading the schema stored as data inside the catalogue. Then run `SELECT COUNT(*)` on a table, insert a row, and count again: the schema stayed, the instance changed.

✓ CHECK YOUR PROGRESS 1.5

1. Which changes more frequently, schema or instance?

Answer: The instance changes with every insert, update or delete. The schema changes only when the design changes, for example through a migration.

2. Where does a DBMS store the schema?

Answer: In the data dictionary or system catalogue, which is itself a set of tables, such as `information_schema` or `pg_catalog`.

1.6 DBMS Architecture: 1-tier, 2-tier and 3-tier

"Tier" here means how the user, the application logic and the database are split across machines or processes.

1.6.1 1-tier architecture

The user works directly on the machine where the DBMS runs. Example: a developer running `psql` on their laptop against a local PostgreSQL, or a mobile app using SQLite on the phone itself. Simple, used for development and local apps. No network in between.

1.6.2 2-tier (client-server) architecture

The client application talks directly to the database server over the network, using an API like ODBC or JDBC. Example: a desktop billing application in a shop that connects straight to a MySQL server in the back office. Simple and fast for a small number of users, but has drawbacks: every client needs DB credentials, business logic lives in each client, and the database must handle connections from every client directly, which does not scale to millions of users.

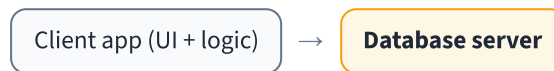


Figure 1.3: 2-tier architecture

1.6.3 3-tier architecture

A middle tier (application server) sits between client and database. The client (browser or mobile app) talks to the application server over HTTP. Only the application server talks to the database. Almost every web application you will work on is 3-tier.

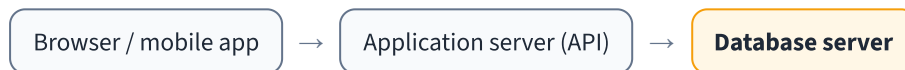


Figure 1.4: 3-tier architecture (presentation, application, data)

Feature	1-tier	2-tier	3-tier
Where logic runs	Same machine as DB	Client	Application server
Security	Local only	Clients hold DB credentials	DB hidden behind app server
Scalability	Single user	Tens to hundreds of users	Very high (scale app servers horizontally)
Maintenance	Easy	Update every client	Update server once
Example	SQLite in a phone app	Shop billing desktop app	Swiggy, IRCTC, any web app

AT WORK

In 3-tier systems the database connection pool lives in the app tier. A common production incident: the team scales the API from 4 pods to 40 pods, each with a pool of 20 connections, and suddenly PostgreSQL sees 800 connections and starts refusing them ("too many clients already"). The fix is a connection pooler like PgBouncer or smaller pools. Tiers are not just a diagram; they decide where your bottlenecks are.

✓ CHECK YOUR PROGRESS 1.6

1. Why is 3-tier architecture more secure than 2-tier?

Answer: In 3-tier, clients never connect to the database directly, so they never hold database credentials. Only the application server, inside a protected network, talks to the database and can validate every request.

2. Give an example of 1-tier architecture.

Answer: A mobile app storing data in a local SQLite file, or a developer using a database client on the same machine as the database.

1.7 Components of a DBMS

Inside a DBMS there are two big parts: the **query processor**, which understands and plans your SQL, and the **storage manager**, which actually reads and writes data on disk safely.

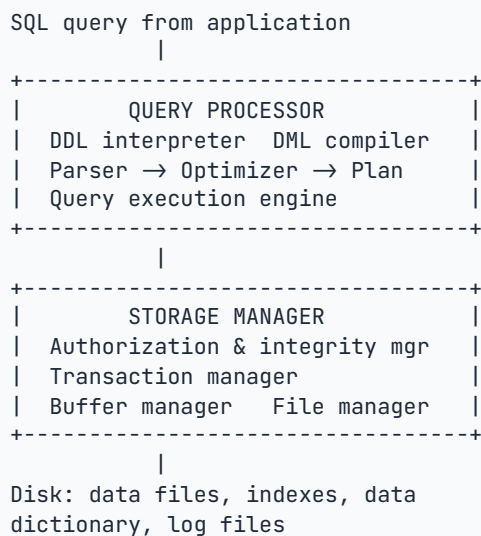


Figure 1.5: Main components of a DBMS

1.7.1 Query processor

- **DDL interpreter:** processes DDL statements (CREATE, ALTER) and records the definitions in the data dictionary.
- **DML compiler / parser:** checks SQL syntax, checks that tables and columns exist, and converts the query into an internal form (a relational algebra tree, see Chapter 4).
- **Query optimizer:** chooses the cheapest way to run the query among many equivalent plans: which index to use, which join order, hash join or nested loop. This is why SQL is **declarative**: you say what you want, the optimizer decides how.
- **Query execution engine:** runs the chosen plan and returns rows.

1.7.2 Storage manager

- **Authorization and integrity manager:** checks privileges and constraints (is this user allowed? does this row violate a CHECK or foreign key?).
- **Transaction manager:** ensures ACID properties. Includes the concurrency control manager (locks, MVCC) and the recovery manager (logs, crash recovery).
- **File manager:** manages allocation of space on disk and the data structures used to store data.
- **Buffer manager:** brings disk pages into main memory and decides which pages to keep or throw out.

1.7.3 Buffer manager in a little more detail

Disk (even SSD) is far slower than RAM. The DBMS therefore keeps a large area of memory called the **buffer pool**, divided into frames, each holding one disk page. When a query needs a page:

1. The buffer manager checks if the page is already in the pool (a **hit**). If yes, it returns it immediately.
2. If not (a **miss**), it picks a free frame. If none is free, it chooses a victim page using a **replacement policy** such as LRU or the clock algorithm.
3. If the victim page is **dirty** (modified in memory), it is written to disk first (respecting the write-ahead logging rule: the log record must reach disk before the data page).
4. The requested page is read into the frame, and it is **pinned** while in use so it is not evicted.

Worked example: a buffer pool has 3 frames and uses LRU. Pages are requested in order A, B, C, A, D, B.

Request	Frames after	Hit/Miss	Evicted
A	A	Miss	-
B	A, B	Miss	-
C	A, B, C	Miss	-
A	A, B, C	Hit	-
D	A, C, D	Miss	B (least recently used)
B	A, D, B	Miss	C

Hit ratio = 1 hit / 6 requests $\approx$ 16.7%. Real databases aim for buffer hit ratios above 99% for hot OLTP data, which is why "give the database more RAM" is such common advice.

INTERVIEW CORNER

Q: What does the query optimizer do?

A: It takes a parsed query and chooses the cheapest execution plan among equivalent ones, using statistics about table sizes and indexes. It decides access paths (index scan vs full scan), join order and join algorithm.

Q: What is a dirty page?

A: A page in the buffer pool that has been modified in memory but not yet written to disk. It must be flushed (after its log records are on disk) before the frame can be reused.

✓ CHECK YOUR PROGRESS 1.7

1. Name the two major components of a DBMS.

Answer: The query processor (DDL interpreter, DML compiler, optimizer, execution engine) and the storage manager (authorization and integrity manager, transaction manager, file manager, buffer manager).

2. Which component guarantees that a committed transaction survives a crash?

Answer: The transaction manager, specifically its recovery manager, using the log (write-ahead logging) to redo committed work after a crash.

3. With 2 frames and LRU, how many misses for the requests A, B, A, C, B?

Answer: A miss, B miss, A hit, C miss (evict B), B miss (evict A). So 4 misses and 1 hit.

1.8 DBMS Languages

SQL is one language, but its commands fall into groups. Interviewers love asking "Is TRUNCATE DDL or DML?" so learn these well.

Group	Full form	Purpose	Commands
DDL	Data Definition Language	Define and change structure (schema)	CREATE, ALTER, DROP, TRUNCATE, RENAME, COMMENT
DML	Data Manipulation Language	Read and change data (instance)	SELECT, INSERT, UPDATE, DELETE, MERGE
DCL	Data Control Language	Permissions	GRANT, REVOKE
TCL	Transaction Control Language	Manage transactions	COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION

Some books put SELECT in a separate group called **DQL** (Data Query Language). Both answers are accepted; mention it to show you know.

```

-- DDL
CREATE TABLE accounts (id INT PRIMARY KEY, owner VARCHAR(50), balance DECIMAL(12,2) CHECK
(balance ≥ 0));
ALTER TABLE accounts ADD COLUMN created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP;

-- DML
INSERT INTO accounts (id, owner, balance) VALUES (1, 'Meera', 1000), (2, 'Arjun', 500);
UPDATE accounts SET balance = balance - 200 WHERE id = 1;

-- TCL
BEGIN;
UPDATE accounts SET balance = balance - 300 WHERE id = 1;
UPDATE accounts SET balance = balance + 300 WHERE id = 2;
COMMIT;

-- DCL
GRANT SELECT ON accounts TO report_user;
REVOKE SELECT ON accounts FROM report_user;

```

1.8.1 DELETE vs TRUNCATE vs DROP

Point	DELETE	TRUNCATE	DROP
Type	DML	DDL	DDL
Removes	Selected rows (WHERE allowed)	All rows	Whole table: data + structure
Row triggers fire?	Yes	No	No
Speed on big tables	Slow (logs each row)	Fast (deallocates pages)	Fast
Rollback	Yes, inside a transaction	Depends on DBMS: yes in PostgreSQL and SQL Server inside a transaction; no in MySQL and Oracle (implicit commit)	Same as TRUNCATE; PostgreSQL allows rollback
Identity/auto-increment	Not reset	Usually reset	Gone with table

⚠️ COMMON MISTAKE

Saying "TRUNCATE can never be rolled back." It depends on the DBMS. In MySQL and Oracle, DDL statements cause an implicit commit, so you cannot roll back. In PostgreSQL, DDL is transactional, so `BEGIN; TRUNCATE t; ROLLBACK;` restores the data. In an interview, say "it depends on the DBMS" and give this example.

✓ CHECK YOUR PROGRESS 1.8

1. **Classify:** GRANT, TRUNCATE, SAVEPOINT, MERGE.

Answer: GRANT is DCL, TRUNCATE is DDL, SAVEPOINT is TCL, MERGE is DML.

2. **Which one keeps the table structure: DELETE without WHERE, TRUNCATE, or DROP?**

Answer: DELETE and TRUNCATE both keep the structure and remove the rows. DROP removes the table itself.

1.9 Database Users and the DBA

1.9.1 Types of database users

- **Naïve (end) users:** use the database through an app without knowing it exists. A passenger booking a Tatkal ticket on IRCTC.
- **Application programmers:** write the code that talks to the database (you, mostly). They use SQL, ORMs like Hibernate, Django ORM, Prisma.
- **Sophisticated users / analysts:** write ad-hoc SQL queries or use BI tools (Metabase, Tableau, Power BI) to analyse data.
- **Database Administrator (DBA):** has central control over the database and the DBMS.

1.9.2 Role of the DBA

1. **Schema definition:** creates the conceptual schema with DDL, often reviewing migrations from developers.
2. **Storage structure and access methods:** decides indexes, partitioning, tablespaces.
3. **Granting authorisation:** creates users and roles, controls who can see or change what.
4. **Routine maintenance:** backups, testing restores, monitoring disk space, applying patches, upgrading versions, vacuuming (PostgreSQL) or optimising tables.
5. **Performance tuning:** reading slow query logs, examining execution plans, tuning memory settings.
6. **High availability and disaster recovery:** setting up replication and failover.

In many modern companies there is no separate DBA. Managed services (Amazon RDS, Google Cloud SQL, Azure SQL) handle backups and patching, and backend developers or SREs handle schema and query tuning. So these DBA skills increasingly become your skills.

💡 DO YOU KNOW?

In January 2017, GitLab.com suffered a well-known outage when an engineer accidentally deleted data on the primary database server during maintenance. GitLab published a detailed public postmortem, which revealed that several of their backup methods were not working as expected, and about six hours of production data was lost. The lesson every DBA repeats: a backup you have never restored is not a backup.

✓ CHECK YOUR PROGRESS 1.9

1. Name any four responsibilities of a DBA.

Answer: Defining the schema, granting and revoking access, taking and testing backups, performance tuning (indexes, configuration), and managing replication and recovery.

2. An IRCTC passenger booking a ticket is which kind of database user?

Answer: A naive or end user, who interacts with the database only through an application interface.

1.10 Types of Databases

1.10.1 Relational databases (RDBMS)

Data is stored in tables (relations) of rows and columns, with a fixed schema, keys linking tables, and SQL as the query language. They offer strong ACID transactions and are the default choice for business data: orders, payments, users, inventory. Examples: PostgreSQL, MySQL, Oracle, SQL Server, SQLite.

1.10.2 NoSQL databases

"NoSQL" (often read as "not only SQL") covers non-relational databases built for flexible schemas, massive scale or special access patterns. Four main families:

Type	Data shape	Good for	Examples
Key-value	key → value	Caching, sessions, counters	Redis, Amazon DynamoDB
Document	JSON-like documents	Catalogues, content, flexible fields	MongoDB, Couchbase
Wide-column	Rows with dynamic columns, grouped by partition key	Huge write-heavy data, time series, messages	Apache Cassandra, HBase
Graph	Nodes and edges	Social networks, recommendations, fraud rings	Neo4j, Amazon Neptune

Point	Relational	NoSQL (typical)
Schema	Fixed, defined up front	Flexible or schema-on-read
Scaling	Mostly vertical; horizontal with effort (read replicas, sharding)	Designed for horizontal scaling
Transactions	Strong ACID across tables	Often limited (single document or key), though many now support multi-document transactions
Joins	Rich joins	Few or none; data is denormalised

Point	Relational	NoSQL (typical)
Query language	SQL	Product-specific APIs

💡 DO YOU KNOW?

Discord's engineering blog describes storing billions of messages first in MongoDB, then moving to Cassandra, and later (in 2023) migrating to ScyllaDB, a Cassandra-compatible database, to reduce latency problems. It is a real-world example of choosing a wide-column store for huge, append-heavy data.

1.10.3 OLTP vs OLAP

OLTP (Online Transaction Processing) systems run the day-to-day business: many small, fast reads and writes, each touching a few rows. Booking a ticket, placing an order, paying via UPI. **OLAP (Online Analytical Processing)** systems answer big analytical questions: few, heavy queries that scan millions of rows to aggregate. "Total sales by city by month for the last 3 years."

Point	OLTP	OLAP
Purpose	Run the business	Analyse the business
Queries	Short, simple, many per second	Long, complex aggregations
Rows touched	Few	Millions
Data	Current, normalised	Historical, often denormalised (star schema)
Storage layout	Row-oriented	Usually column-oriented
Users	Customers, app servers	Analysts, data scientists, dashboards
Examples	PostgreSQL, MySQL, Oracle	Snowflake, Google BigQuery, Amazon Redshift, ClickHouse

Data typically flows from OLTP to OLAP through an **ETL** or **ELT** pipeline (Extract, Transform, Load) into a **data warehouse**.



Figure 1.6: OLTP data flowing into an OLAP warehouse

⚠️ COMMON MISTAKE

Running a heavy analytics query ("revenue by month for 3 years") directly on the production OLTP primary database during peak hours. It eats CPU and I/O and slows every checkout. Run such reports on a read replica or, better, in a warehouse.

 THINK AND DISCUSS

A food delivery app stores orders, restaurant menus, delivery partner GPS pings every 5 seconds, and user sessions. Which database type would you pick for each, and would you use one database or several? What do you lose by using several?

 CHECK YOUR PROGRESS 1.10

1. Name the four families of NoSQL databases with one example each.

Answer: Key-value (Redis), document (MongoDB), wide-column (Cassandra), graph (Neo4j).

2. Why do OLAP databases usually store data column-wise?

Answer: Analytical queries read a few columns across millions of rows. Column storage reads only the needed columns and compresses similar values well, so scans and aggregations are much faster.

3. Is a UPI payment an OLTP or OLAP operation?

Answer: OLTP. It is a short transaction touching a few rows that must be fast and fully ACID.

1.11 Popular DBMS Products

Product	Type	Licence	Known for
Oracle Database	Relational	Commercial	Large enterprises, banks, PL/SQL
MySQL	Relational	Open source (GPL) + commercial editions	Web applications, LAMP stack; owned by Oracle
PostgreSQL	Relational (object-relational)	Open source (PostgreSQL licence)	Standards compliance, extensions (PostGIS, JSONB)
Microsoft SQL Server	Relational	Commercial (free Express edition)	Microsoft / .NET shops, T-SQL
SQLite	Relational, embedded	Public domain	Mobile apps, browsers, small tools
MongoDB	Document	Source-available (SSPL)	Flexible JSON documents
Redis	Key-value, in-memory	Licence has changed over time; check current terms	Caching, rate limits, leaderboards
Cassandra	Wide-column	Open source (Apache)	Massive write throughput, multi-datacenter
Neo4j	Graph	Community + commercial	Relationships, Cypher query language
Snowflake, BigQuery	Analytical (OLAP) warehouses	Commercial cloud services	Petabyte-scale analytics

 ACTIVITY 1.2

Install SQLite (or use an online SQLite playground) and create two tables: `students(roll_no, name, city)` and `fees(roll_no, due)`. Insert 5 rows. Now write the "students from Pune with pending fees" query using a JOIN. Then imagine writing the same thing with two CSV files and a Python script. Count the lines of code in each approach.

 INTERVIEW CORNER

Q: When would you choose MongoDB over PostgreSQL?

A: When the data is naturally document-shaped with varying fields, is read and written mostly as a whole document, and needs easy horizontal scaling, for example a product catalogue with different attributes per category. For money, inventory and strongly related data needing joins and multi-row transactions, PostgreSQL is usually the safer default (and PostgreSQL's JSONB also covers many flexible-field needs).

Q: What is the difference between OLTP and OLAP?

A: OLTP handles many short transactions on current data with row storage and normalised schemas. OLAP handles few long analytical queries over historical data, usually with column storage and denormalised star schemas.

 CHECK YOUR PROGRESS 1.11

1. Which popular DBMS runs inside the application process without a server?

Answer: SQLite. It is an embedded, serverless DBMS stored in a single file.

2. Name two cloud data warehouses used for OLAP.

Answer: Snowflake and Google BigQuery (Amazon Redshift is another).

What Have We Learnt?

- Data is raw facts; information is processed, meaningful data. A database is the data; a DBMS is the software that manages it.
- File-based systems suffer from redundancy, inconsistency, difficult access, isolation, integrity, atomicity, concurrency and security problems.
- A DBMS provides shared data, SQL, constraints, ACID transactions, concurrency control, security and recovery, at the cost of complexity, overhead and money.
- The three-schema architecture has internal (how stored), conceptual (what stored) and external (user views) levels, linked by two mappings.
- Physical data independence: change storage without changing the logical schema. Logical data independence: change the logical schema without changing views and apps. Logical is harder.
- A schema is the structure (intension); an instance is the data at a moment (extension).
- 1-tier: user on the DB machine; 2-tier: client talks directly to DB; 3-tier: app server in the middle, used by most web apps.
- The query processor parses, optimises and executes queries; the storage manager handles authorization, integrity, transactions, files and buffers.
- The buffer manager caches disk pages in memory, using policies like LRU and flushing dirty pages under write-ahead logging.
- SQL commands: DDL (CREATE, ALTER, DROP, TRUNCATE), DML (SELECT, INSERT, UPDATE, DELETE), DCL (GRANT, REVOKE), TCL (COMMIT, ROLLBACK, SAVEPOINT).
- Users include naive users, application programmers, analysts, specialised users and the DBA, who handles schema, security, backups and tuning.
- NoSQL families are key-value, document, wide-column and graph. OLTP runs the business; OLAP analyses it.

Key terms: data, information, database, DBMS, file processing system, redundancy, inconsistency, data isolation, integrity, atomicity, concurrency, three-schema architecture, internal level, conceptual level, external level, view, mapping, physical data independence, logical data independence, schema, instance, data dictionary, 2-tier, 3-tier, query processor, query optimizer, storage manager, buffer manager, buffer pool, dirty page, LRU, DDL, DML, DCL, TCL, DBA, NoSQL, key-value, document, wide-column, graph database, OLTP, OLAP, ETL, data warehouse.

Exercises

A. Multiple Choice Questions

1. Which of these is a DBMS?
 - a. Excel file
 - b. PostgreSQL
 - c. A CSV file
 - d. JSON
2. Two copies of a student's phone number disagree. This is called:
 - a. Data isolation
 - b. Data inconsistency
 - c. Data independence
 - d. Atomicity
3. Which level of the three-schema architecture describes indexes and file organisation?
 - a. External
 - b. Conceptual
 - c. Internal
 - d. View
4. Adding a column to a table without breaking existing views is an example of:
 - a. Physical data independence
 - b. Logical data independence
 - c. Atomicity
 - d. Redundancy
5. TRUNCATE belongs to:
 - a. DML
 - b. DCL
 - c. TCL
 - d. DDL
6. Which component chooses between an index scan and a full table scan?
 - a. Buffer manager
 - b. Query optimizer
 - c. File manager
 - d. DDL interpreter
7. In a 3-tier architecture, the browser talks to:
 - a. The database directly
 - b. The application server
 - c. The file manager
 - d. The buffer pool
8. Redis is best described as a:
 - a. Graph database
 - b. Column store warehouse
 - c. In-memory key-value store
 - d. Relational DBMS
9. Which workload is typical of OLAP?
 - a. Insert one order
 - b. Update one user's address
 - c. Total sales by region for 5 years
 - d. Check one account balance
10. The actual data in a database at a moment is called the:
 - a. Schema
 - b. Instance
 - c. Intension
 - d. Catalogue

B. Fill in the Blanks

1. Processed, meaningful data is called _____.
2. The three-schema architecture was proposed by the _____ committee.
3. A page modified in memory but not yet written to disk is called a _____ page.
4. GRANT and REVOKE are _____ commands.
5. The schema is stored in the _____, also called the system catalogue.
6. Moving data from OLTP systems to a warehouse is usually done by an _____ pipeline.

C. True or False

1. A database can have many conceptual schemas.
2. Physical data independence is generally easier to achieve than logical data independence.
3. DELETE is a DDL command.
4. In 2-tier architecture, the client connects directly to the database server.
5. OLAP systems commonly use column-oriented storage.

D. Short Answer Questions

1. Differentiate between data and information with an example.
2. What is the atomicity problem in file systems? How does a DBMS solve it?
3. Define schema and instance. Which one changes more often?
4. List the differences between DELETE, TRUNCATE and DROP.
5. What does the buffer manager do?
6. Why is 3-tier architecture preferred for web applications?

E. Long Answer Questions

1. Explain the problems of the file processing system and how a DBMS addresses each one.
2. Explain the three-schema architecture with a neat diagram. Define physical and logical data independence with examples.
3. Describe the components of a DBMS (query processor and storage manager) and the flow of a query through them.
4. Compare relational and NoSQL databases, and OLTP and OLAP systems. Give examples of products for each.

F. Design Challenge

A small coaching institute with 3 branches keeps student records, fee payments and test scores in separate Excel files per branch. Students can attend classes at any branch. Propose a database solution: which type of DBMS, which architecture (tiers), what the main tables are, which users need which views (external schemas), and how you would protect fee data from teachers.

Answer Key: Chapter 1

A. MCQ

1. (b) PostgreSQL is DBMS software; the others are file formats.
2. (b) Disagreeing copies are inconsistency.
3. (c) The internal level describes physical storage.
4. (b) Conceptual schema changed, views unaffected.
5. (d) TRUNCATE is DDL.
6. (b) The optimizer chooses the plan.
7. (b) Only the app server talks to the database.
8. (c) Redis is an in-memory key-value store.
9. (c) Large aggregation over history is OLAP.
10. (b) Instance is the data at a moment.

B. Fill in the Blanks

1. information
2. ANSI/SPARC
3. dirty
4. DCL
5. data dictionary
6. ETL (or ELT)

C. True or False

1. False. There is exactly one conceptual schema; there can be many external schemas.
2. True. Storage changes are fully hidden by the DBMS, while apps depend on logical structure.
3. False. DELETE is DML; it changes data, not structure.
4. True. That is the definition of 2-tier client-server.
5. True. Column storage suits scans of few columns over many rows.

D. Short Answers

1. Data is raw facts without context, such as "45, 78, 90". Information is data processed to give meaning, such as "the class average is 71". Information supports decisions; data alone does not.
2. When an operation needs several writes (debit one account, credit another), a crash in between leaves data half-updated, and files cannot undo it. A DBMS groups the writes into a transaction and uses a log: on crash, uncommitted changes are rolled back, so the operation is all or nothing.
3. A schema is the structure of the database (tables, columns, types, constraints). An instance is the actual set of rows at a given moment. The instance changes with every insert, update or delete; the schema changes only through design changes.
4. DELETE is DML, removes chosen rows with WHERE, fires triggers, logs each row and is slow on big tables. TRUNCATE is DDL, removes all rows quickly, does not fire row triggers and usually resets identity. DROP is DDL and removes the table structure along with its data, indexes and constraints. Rollback of TRUNCATE and DROP depends on the DBMS.

5. The buffer manager moves pages between disk and the in-memory buffer pool. It serves hits from memory, loads pages on misses, chooses victims using a policy like LRU, writes dirty pages back to disk (after their log records), and pins pages in use.
6. In 3-tier, clients never hold database credentials, business logic is in one place and easy to update, and app servers can be scaled horizontally. The database is protected inside a private network, and connection pooling is managed centrally.

E. Long Answers

1. Model answer points: redundancy (same data in many files) solved by a shared, normalised database; inconsistency (copies disagree) solved by single source of truth; difficult access solved by SQL; isolation (many formats) solved by one data model and catalogue; integrity rules hidden in code solved by declarative constraints; atomicity solved by transactions and logging; concurrency anomalies such as lost update solved by locking or MVCC; security solved by users, roles, GRANT/REVOKE and views; plus backup and recovery tools. Mention the cost: complexity and overhead.
2. Model answer points: draw three levels with views at top, conceptual schema in middle, internal schema at bottom; external level gives per-user views; conceptual level describes all entities, attributes, relationships and constraints; internal level describes files, indexes, page layout; two mappings connect levels; physical data independence: change internal schema (add index, partition) without affecting conceptual schema; logical data independence: change conceptual schema (add column, split table) without affecting views, achieved using views; logical is harder.
3. Model answer points: query processor includes DDL interpreter (writes definitions to the dictionary), DML compiler/parser (syntax and semantic checks, builds a relational algebra tree), optimizer (chooses cheapest plan using statistics), execution engine (runs plan); storage manager includes authorization and integrity manager, transaction manager (concurrency control, recovery), file manager, buffer manager; flow: SQL → parse → validate against catalogue → optimise → execute → storage manager fetches pages via buffer pool → rows returned; changes are logged before pages are flushed.
4. Model answer points: relational has fixed schema, tables, SQL, joins and strong ACID (PostgreSQL, MySQL, Oracle); NoSQL has flexible schema, horizontal scaling, limited joins, four families (Redis, MongoDB, Cassandra, Neo4j); choose based on data shape and consistency needs; OLTP handles many short transactions on current normalised data with row storage (PostgreSQL, MySQL); OLAP handles complex aggregations over historical data with column storage and star schemas (Snowflake, BigQuery, Redshift, ClickHouse); data moves via ETL into a warehouse.

F. Design Challenge: Model Solution

Use a relational DBMS (for example PostgreSQL on a managed cloud service), because fees involve money and need ACID transactions and constraints. Use 3-tier architecture: a web app for branch staff, an API server, and the database, so no branch machine stores database passwords. Main tables: `branches(branch_id, name, city)`, `students(student_id, name, phone, home_branch_id)`, `enrolments(student_id, course_id, branch_id)`, `fee_payments(payment_id, student_id, amount CHECK (amount > 0), paid_on, branch_id)`, `tests(test_id, course_id, test_date)`, `scores(student_id, test_id, marks CHECK (marks BETWEEN 0 AND 100))`. One student row is shared by all branches, removing redundancy and inconsistency. External schemas: a `teacher_view` with student names and scores only; an `accounts_view` with names, phones and fee details; a

management view with aggregates per branch. Create roles `teacher` , `accountant` , `manager` ; GRANT `SELECT` on the right views to each and give teachers no privileges on `fee_payments` . Take automated daily backups with point-in-time recovery, and test a restore every month.

CHAPTER 2

Data Modelling and the ER Model

"Weeks of coding can save you hours of drawing an ER diagram."

After studying this chapter, you will be able to

- Compare the hierarchical, network, relational, object and document data models.
- Identify entities, attributes (simple, composite, multi-valued, derived, key) and relationships in a problem statement.
- State the degree, cardinality (1:1, 1:N, M:N) and participation (total, partial) of a relationship.
- Model weak entities, generalization, specialization and aggregation.
- Convert any ER diagram into relational tables using standard mapping rules.
- Spot common data-modelling mistakes in real projects.

Before an architect builds a house, she draws a plan. Nobody pours concrete first and then asks where the kitchen should go. Yet many software teams do exactly that with databases: they create a `users` table on day one, then keep adding columns like `address2`, `address3`, `phone_new` until the table looks like a family WhatsApp group: noisy, repetitive and full of things nobody remembers adding.

Data modelling is the plan for your database. The Entity-Relationship (ER) model, proposed by Peter Chen in 1976, is the most widely taught way to draw that plan. In this chapter we learn to read a problem statement, find the "things" and "links" in it, draw them, and convert the drawing into tables. Interviewers ask "design the schema for Zomato / a library / BookMyShow" all the time, and this chapter is the toolkit for that question.

2.1 Data Models

A **data model** is a collection of concepts used to describe the structure of a database: what data types exist, how they relate, and what constraints apply. Data models are often classified as:

- **Conceptual (high-level) models:** close to how users think. Example: the ER model.
- **Logical (representational) models:** how the data is organised for a DBMS. Example: the relational model (tables).
- **Physical (low-level) models:** how data is stored on disk: files, pages, indexes.



Figure 2.1: Steps of database design

2.1.1 Hierarchical model

Data is organised as a tree. Each child record has exactly one parent. Example: Company → Departments → Employees. Fast for walking down the tree, but many-to-many relationships (a student in many courses, a course with many students) need duplicate records. IBM's IMS, built in the 1960s for the Apollo programme, is the famous example. The Windows Registry and XML documents are tree-shaped too.

2.1.2 Network model

A generalisation of the hierarchical model: records are linked as a graph, so a record can have many parents. Standardised by CODASYL around 1969 to 1971. It handles M:N relationships, but programs must navigate pointers record by record ("find owner, then get next member"), so code is tightly tied to the structure.

2.1.3 Relational model

Data is stored in tables (relations). Relationships are expressed through matching values (foreign keys), not pointers. Queries are declarative (SQL). Proposed by Codd in 1970, it dominates business software today. Chapter 3 covers it in depth.

2.1.4 Object and object-relational models

The object model stores objects with attributes and methods, with inheritance, matching object-oriented languages. Pure object databases (like db4o, ObjectDB) stayed niche. Instead, relational databases added object features, giving **object-relational** systems: PostgreSQL supports user-defined types, arrays and table inheritance. ORMs (Hibernate, Django ORM, Entity Framework) are the everyday bridge between objects in code and rows in tables.

2.1.5 Document model

Data is stored as self-contained documents, usually JSON or BSON, grouped in collections. Each document can have its own fields and nested arrays. Example: MongoDB. Great when data is read and written as a whole unit (a product with all its specs). Weak when you need many cross-document relationships and joins.

```
// A document in MongoDB: an order with its items embedded
{
  "_id": 1001,
  "customer": { "id": 7, "name": "Ananya" },
  "items": [
    { "sku": "TEA-500", "qty": 2, "price": 180 },
    { "sku": "MUG-01", "qty": 1, "price": 250 }
  ],
  "status": "SHIPPED"
}
```

Model	Structure	Links via	Strength	Weakness	Example
Hierarchical	Tree	Parent pointers	Fast tree traversal	M:N needs duplication	IBM IMS
Network	Graph of records	Pointers (sets)	Handles M:N	Navigational, complex code	IDMS
Relational	Tables	Key values	Simple, declarative SQL, strong theory	Object mismatch, joins at scale	PostgreSQL, MySQL
Object	Objects, classes	Object references	Matches OOP code	Niche, weak query standards	ObjectDB
Document	JSON documents	Embedding or references	Flexible schema, whole-object reads	Weak joins, duplication	MongoDB

💡 DO YOU KNOW?

IBM's IMS (Information Management System) was developed in the mid-1960s with Rockwell and Caterpillar to track the bill of materials for the Apollo space programme. Versions of IMS are still in use at large banks today.

✅ CHECK YOUR PROGRESS 2.1

1. Why is the hierarchical model poor at representing many-to-many relationships?

Answer: Each child can have only one parent, so a record shared by many parents (a course taken by many students) has to be duplicated under each parent, causing redundancy.

2. How does the relational model link related data, compared with the network model?

Answer: The relational model links data through matching key values (foreign keys). The network model uses physical pointers that programs must navigate one record at a time.

2.2 Entities and Attributes

2.2.1 Entity and entity set

An **entity** is a real-world thing or object that is distinguishable from others and about which we want to store data. It can be physical (a student, a train, a product) or conceptual (a course, a booking, a loan). An **entity set** (or entity type) is a collection of entities of the same type sharing the same attributes: all students form the `STUDENT` entity set. In tables: entity set $\approx$ table, entity $\approx$ row.

Quick trick for finding entities in a problem statement: **nouns** are candidate entities or attributes, **verbs** are candidate relationships. "A customer *places* orders. Each order *contains* products." Entities: Customer, Order, Product. Relationships: places, contains.

2.2.2 Types of attributes

An **attribute** is a property describing an entity. The set of allowed values is the attribute's **domain**.

Type	Meaning	Example	ER notation
Simple (atomic)	Cannot be divided further	roll_no, age	Oval
Composite	Made of sub-parts	name = first + last; address = street + city + pin	Oval with child ovals
Single-valued	One value per entity	date_of_birth	Oval
Multi-valued	Many values per entity	phone_numbers, skills	Double oval
Derived	Computed from other attributes	age (from date_of_birth), order_total	Dashed oval
Stored	Actually stored, source of derived	date_of_birth	Oval
Key	Uniquely identifies an entity	roll_no, aadhaar_no	Oval with underlined name
Complex	Composite + multi-valued together	Several addresses, each with street, city, pin	Double oval with child ovals

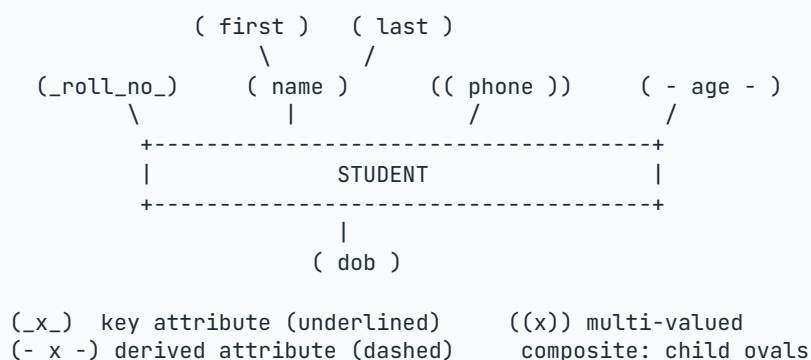


Figure 2.2: Attribute types around the STUDENT entity

⚠ COMMON MISTAKE

Storing age as a column. Age is derived; it changes every year without anyone updating it. Store `date_of_birth` and compute age in the query. The same applies to "order_total" in many cases, though at work it is sometimes stored deliberately for speed and auditing (you then must keep it in sync carefully).

✏ ACTIVITY 2.1

Take your own Aadhaar, PAN card or college ID card. List every field on it and classify each as simple/composite, single/multi-valued, stored/derived, and mark the key attribute.

✅ CHECK YOUR PROGRESS 2.2

1. Classify "address" with street, city and pin, where a person can have several addresses.

Answer: It is a complex attribute: composite (street, city, pin) and multi-valued (several addresses).

2. What is the difference between an entity and an entity set?

Answer: An entity is one distinguishable object, like student Riya. An entity set is the collection of all entities of that type, like all students. In a table, an entity is a row and the entity set is the table.

3. How is a derived attribute shown in an ER diagram?

Answer: With a dashed oval.

2.3 Relationships, Degree, Cardinality and Participation

2.3.1 Relationship and relationship set

A **relationship** is an association among entities: "Riya ENROLLS IN DBMS course". A **relationship set** is the set of all such associations of the same type. Relationships can have their own attributes, called **descriptive attributes**: `ENROLLS(grade, enrolled_on)`. The grade belongs neither to the student alone nor to the course alone; it belongs to the pair.

Notation (Chen): entity = rectangle, relationship = diamond, attribute = oval, lines connect them.

2.3.2 Degree of a relationship

The **degree** is the number of entity sets that take part in a relationship.

- **Unary (recursive), degree 1:** an entity set related to itself. EMPLOYEE *manages* EMPLOYEE. Each participating entity plays a **role**: manager and subordinate.
- **Binary, degree 2:** two entity sets. STUDENT *enrolls* COURSE. The most common case.
- **Ternary, degree 3:** three entity sets. DOCTOR *prescribes* MEDICINE to PATIENT.

- **n-ary**: n entity sets; rare in practice.

🤔 THINK AND DISCUSS

Can the ternary relationship "DOCTOR prescribes MEDICINE to PATIENT" be replaced by three binary relationships (doctor-patient, doctor-medicine, patient-medicine)? Try an example: Dr A treats P1 and P2, prescribed paracetamol to P1 and cough syrup to P2. Do the three binary tables still tell you who prescribed what to whom?

2.3.3 Cardinality (mapping cardinality)

Cardinality tells how many entities of one set can be associated with how many entities of the other set, for a binary relationship.

Cardinality	Meaning	Example
One-to-one (1:1)	An A relates to at most one B, and a B to at most one A	Person HAS Passport (one valid passport each); Employee HEADS Department
One-to-many (1:N)	An A relates to many B; each B to at most one A	Department HAS Employees; Customer PLACES Orders
Many-to-one (N:1)	Same as 1:N viewed from the other side	Employees WORK IN a Department
Many-to-many (M:N)	An A relates to many B, and a B to many A	Students ENROLL IN Courses; Actors ACT IN Movies

How to decide: ask two questions, one in each direction. "Can one customer place many orders?" Yes. "Can one order be placed by many customers?" No. So Customer:Order is 1:N. Always ask both directions; most modelling bugs come from asking only one.

2.3.4 Participation constraints

- **Total participation**: every entity in the set must take part in at least one relationship instance. Every ORDER must be placed by some CUSTOMER. Shown by a **double line**.
- **Partial participation**: some entities may not take part. Not every CUSTOMER has placed an order (a new sign-up). Shown by a single line.

Cardinality gives the maximum; participation gives the minimum. Together they are often written in **(min, max)** notation: ORDER participates in PLACES with (1,1) meaning exactly one customer; CUSTOMER participates with (0,N) meaning zero or more orders.

```
+-----+ 1          N +-----+
| CUSTOMER |-----<PLACES>====| ORDER |
+-----+          +-----+
partial (single line)      total (double line ====)
```

Read: one customer places many orders; every order must belong to exactly one customer.

Figure 2.3: Cardinality and participation in Chen notation

2.3.5 Crow's foot notation

In industry tools (dbdiagram.io, MySQL Workbench, Lucidchart, draw.io) you will mostly see **crow's foot** notation instead of Chen's diamonds. Relationships are lines between tables, and the line ends show cardinality:

```
--| - exactly one      --|← one or many
--o| - zero or one     --o← zero or many

CUSTOMER --|-----o← ORDER
(each order has exactly one customer;
 a customer has zero or many orders)
```

Figure 2.4: Crow's foot symbols

🤪 FUNNY SIDE

Cardinality in Indian family terms: Mother to child is 1:N. Child to biological mother is N:1. Cousins to cousins is M:N. Wedding guest list to relatives is "M:N, total participation, and nobody is allowed to be partial or there will be a complaint for ten years."

🔗 INTERVIEW CORNER

Q: What is the difference between cardinality and participation?

A: Cardinality is the maximum number of relationship instances an entity can take part in (1 or many). Participation is the minimum: total means every entity must participate (at least one), partial means participation is optional.

Q: Give an example of a recursive relationship and how you store it.

A: Employee manages employee. Store it as a self-referencing foreign key: `employees(id, name, manager_id REFERENCES employees(id))`, with `manager_id` NULL for the CEO.

✅ CHECK YOUR PROGRESS 2.3

1. What is the cardinality between Movie and Actor? Explain using two questions.

Answer: One movie has many actors, and one actor acts in many movies, so it is M:N.

2. Every loan must belong to some customer. Which participation is this, and how is it drawn?

Answer: Total participation of LOAN in the relationship, drawn as a double line between LOAN and the relationship diamond.

3. What is the degree of "SUPPLIER supplies PART to PROJECT"?

Answer: 3 (ternary), because three entity sets participate.

2.4 Weak Entities and Identifying Relationships

A **strong entity** has a key of its own. A **weak entity** does not have enough attributes to form a key by itself; it can be identified only together with another entity, called its **owner** or **identifying entity**.

Example: a bank LOAN has PAYMENTs numbered 1, 2, 3... Payment number 3 exists for thousands of loans. So `payment_no` alone is not unique. It is unique only within a loan: (loan_id, payment_no). PAYMENT is a weak entity, LOAN is the owner.

- The attribute that distinguishes weak entities of the same owner is the **partial key** (discriminator), shown with a **dashed underline**: `payment_no` .
- The relationship linking the weak entity to its owner is the **identifying relationship**, drawn as a **double diamond**.
- The weak entity is drawn as a **double rectangle** and always has **total participation** in the identifying relationship.
- The owner to weak entity relationship is usually 1:N.
- Key of weak entity = owner's primary key + partial key.

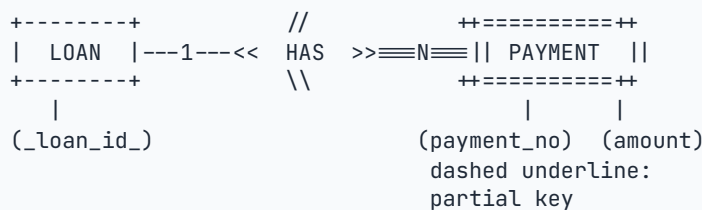


Figure 2.5: PAYMENT is a weak entity owned by LOAN (double rectangle, double diamond)

More examples: **DEPENDENT** of an **EMPLOYEE** (identified by `employee_id` + dependent name), **ROOM** in a **HOTEL** (room 101 exists in every hotel), **SEAT** in a **SHOW** (seat A5 exists in every show), **ORDER LINE** in an **ORDER** (line 1, line 2).

 COMMON MISTAKE

Thinking "weak" means "unimportant". A weak entity can be very important (payments!). It is weak only in identification: its existence depends on the owner. If the loan is deleted, its payments make no sense, which is why weak entity tables usually have `ON DELETE CASCADE` foreign keys to the owner.

AT WORK

Many teams give every table a surrogate key (`id BIGSERIAL`), including `ORDER_LINES`. Then the weak-entity nature becomes invisible, and duplicates sneak in: the same order gets two "line 1" rows after a retried API call. The fix in code review: keep the surrogate id if you like, but also add `UNIQUE (order_id, line_no)`, which is exactly the weak entity's real key.

✓ CHECK YOUR PROGRESS 2.4

1. What is the primary key of a weak entity?

Answer: The primary key of its owner (identifying) entity combined with its own partial key (discriminator).

2. Why must a weak entity have total participation in its identifying relationship?

Answer: Because it cannot be identified without its owner. Every weak entity must be linked to exactly one owner, otherwise it has no key.

2.5 Extended ER (EER) Features

The basic ER model cannot express "a car and a bike are both vehicles" or "a relationship that itself takes part in another relationship". The **Extended (Enhanced) ER model** adds these.

2.5.1 Specialization (top-down)

Specialization starts with a general entity and defines sub-classes with extra attributes. EMPLOYEE is specialised into FULL_TIME (salary, pf_number) and CONTRACTOR (hourly_rate, agency). All employees share emp_id, name; each subclass has its own extras. Shown with an "ISA" triangle (or a circle in some textbooks).

2.5.2 Generalization (bottom-up)

Generalization is the reverse: notice that CAR and BIKE share attributes (reg_no, owner, model) and create a general super-class VEHICLE. Same diagram as specialization; the difference is the direction of thinking.

2.5.3 Inheritance and constraints

Sub-classes **inherit** all attributes and relationships of the super-class. A FULL_TIME employee has emp_id and name from EMPLOYEE and also takes part in any relationship EMPLOYEE has (WORKS_IN department).

- **Disjoint vs overlapping:** disjoint means an entity belongs to at most one subclass (an employee is either full-time or contractor). Overlapping means it can be in several (a PERSON can be both STUDENT and EMPLOYEE, like a working MBA student).
- **Total vs partial (completeness):** total means every super-class entity must be in some subclass (every account is SAVINGS or CURRENT). Partial means some may be in none (some vehicles are neither car nor bike, like a tractor).

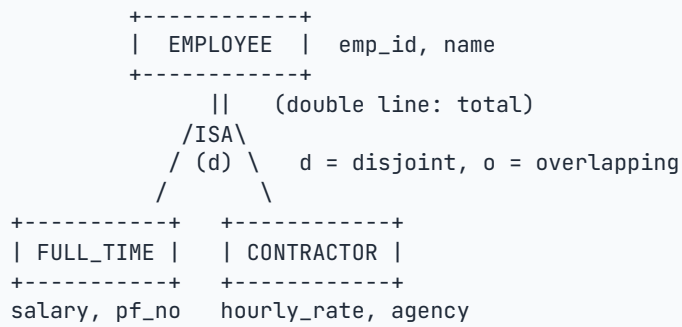


Figure 2.6: Specialization of EMPLOYEE (total, disjoint)

2.5.4 Aggregation

The ER model does not allow a relationship to take part in another relationship. **Aggregation** treats a relationship (with its entities) as a higher-level entity so it can participate in another relationship.

Example: EMPLOYEE *works on* PROJECT. A MANAGER *monitors* each (employee, project) assignment, not the employee alone or the project alone. We draw a box around WORKS_ON with EMPLOYEE and PROJECT, and connect MONITORS to that box.



Figure 2.7: Aggregation: MONITORS relates MANAGER to the WORKS_ON relationship

Concept	Direction	Purpose	Example
Specialization	Top-down	Define subclasses with extra features	ACCOUNT → SAVINGS, CURRENT
Generalization	Bottom-up	Combine common features into a superclass	CAR, BIKE → VEHICLE
Inheritance	Superclass → subclass	Subclasses get attributes and relationships	SAVINGS has acc_no from ACCOUNT
Aggregation	Relationship as entity	Let a relationship join another relationship	MANAGER monitors (EMPLOYEE works on PROJECT)

✓ CHECK YOUR PROGRESS 2.5

1. How do generalization and specialization differ?

Answer: Specialization is top-down: start from a general entity and split it into subclasses. Generalization is bottom-up: combine entities with common attributes into a superclass. The resulting diagram looks the same.

2. A person can be both a customer and an employee of a bank. Is this disjoint or overlapping?

Answer: Overlapping, because one entity can belong to more than one subclass.

3. When do you need aggregation?

Answer: When a relationship must participate in another relationship, for example a manager monitoring a particular employee-project assignment.

2.6 ER Diagram Notation Summary

Symbol (Chen)	ASCII used in this book	Meaning
Rectangle	+-----+ box	Strong entity set
Double rectangle	++=====++ box	Weak entity set
Diamond	<NAME>	Relationship set
Double diamond	<<NAME>>	Identifying relationship
Oval	(attr)	Attribute
Underlined oval	(_attr_)	Key attribute
Dashed-underlined oval	marked "partial key"	Partial key of weak entity
Double oval	((attr))	Multi-valued attribute
Dashed oval	(- attr -)	Derived attribute
Double line	=====	Total participation
Triangle "ISA"	/ISA\	Specialization / generalization
Box around relationship	+=====+ enclosing	Aggregation

✏ ACTIVITY 2.2

Open dbdiagram.io (free, in the browser) and type this DBML: `Table customers { id int [pk] name varchar } and Table orders { id int [pk] customer_id int [ref: > customers.id] } .` Watch it draw a crow's foot diagram. Now add a `products` table and an `order_items` table to model the M:N between orders and products.

✓ CHECK YOUR PROGRESS 2.6

1. Which symbols indicate a weak entity and its identifying relationship?

Answer: A double rectangle for the weak entity and a double diamond for the identifying relationship.

2. In crow's foot notation, what does `o<` at a line end mean?

Answer: Zero or many: optional participation with a maximum of many.

2.7 Converting an ER Diagram to Tables

This is the most asked practical skill in this chapter. Learn the rules, then practise on the worked example.

2.7.1 Mapping rules

1. **Strong entity:** make a table with all its simple attributes. The key attribute becomes the primary key.
2. **Composite attribute:** store only its simple components as columns (`first_name` , `last_name`), not the composite itself.
3. **Multi-valued attribute:** make a separate table with the owner's primary key plus the attribute. Primary key = both columns. Example: `student_phones(roll_no, phone)` .
4. **Derived attribute:** usually do not store it; compute it (or store with care for performance).
5. **Weak entity:** make a table with its attributes plus the owner's primary key as a foreign key. Primary key = owner PK + partial key. Use `ON DELETE CASCADE` .
6. **1:1 relationship:** put the primary key of one side as a foreign key (with UNIQUE) in the other. Prefer the side with total participation, so the column is never NULL. If both sides are total, you may merge into one table.
7. **1:N relationship:** put the primary key of the "1" side as a foreign key in the "N" side table. Relationship attributes also go to the N side. No new table.
8. **M:N relationship:** create a new table (junction / bridge / associative table) with foreign keys to both sides. Primary key = both foreign keys combined (or a surrogate id plus a UNIQUE constraint). Relationship attributes go here.
9. **Unary (recursive) relationship:** 1:N becomes a self-referencing foreign key (`manager_id`). M:N becomes a new table with two foreign keys to the same table (`friendships(user_id, friend_id)`).
10. **n-ary relationship:** create a table with foreign keys to all participating entities; the primary key is usually the combination (minus any side with cardinality 1).
11. **Specialization / generalization:** three common options, see 2.7.3.
12. **Aggregation:** the relationship inside the aggregation becomes a table (as per its cardinality); the outer relationship references that table's primary key.

 KEY POINT

Memorise: 1:N → **foreign key on the many side**. M:N → **new junction table**. Multi-valued → **new table**. Weak entity → **owner key + partial key**. These four lines answer most schema-design interview questions.

2.7.2 Worked example: a college database

Requirements:

- A college has departments, each with a dept_id, name and building.
- Each student has roll_no, name (first, last), date of birth and one or more phone numbers. Every student belongs to exactly one department; a department has many students.
- Each professor has prof_id, name and salary, and works in one department. Each department is headed by exactly one professor, and a professor heads at most one department.
- Courses have course_code, title and credits, and are offered by one department.
- Students enroll in many courses; a course has many students. Each enrolment records the semester and grade.
- A professor can mentor other professors (a junior has at most one mentor).
- Each course has sections numbered 1, 2, 3 within the course, each with a room.

Step 1: find entities and attributes. DEPARTMENT(dept_id, name, building). STUDENT(roll_no, name{first,last}, dob, {phones}, age derived). PROFESSOR(prof_id, name, salary). COURSE(course_code, title, credits). SECTION(section_no, room) is weak, owned by COURSE.

Step 2: find relationships with cardinality and participation.

Relationship	Entities	Cardinality	Participation
BELONGS_TO	Student - Department	N:1	Student total
WORKS_IN	Professor - Department	N:1	Professor total
HEADS	Professor - Department	1:1	Department total, Professor partial
OFFERS	Department - Course	1:N	Course total
ENROLLS	Student - Course (attrs: semester, grade)	M:N	Partial both
MENTORS	Professor - Professor	1:N (recursive)	Partial
HAS_SECTION	Course - Section	1:N identifying	Section total

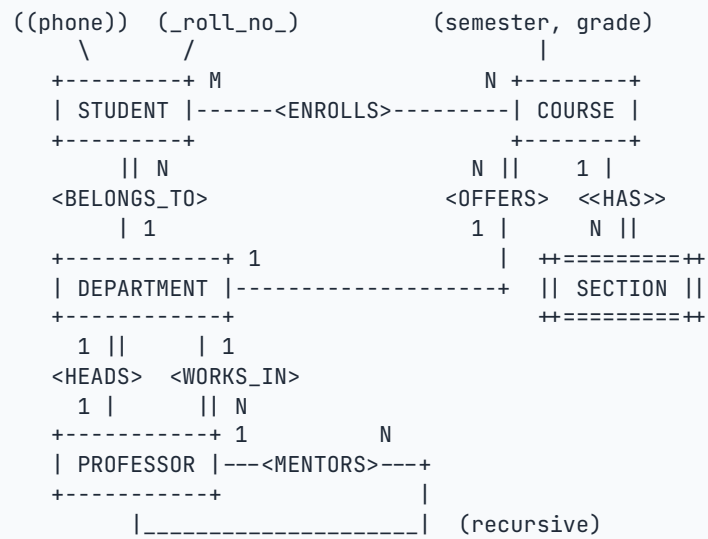


Figure 2.8: ER sketch of the college database (double lines show total participation)

Step 3: apply the mapping rules.

```

CREATE TABLE departments (
  dept_id    INT PRIMARY KEY,
  name       VARCHAR(60) NOT NULL UNIQUE,
  building   VARCHAR(40),
  head_prof_id INT UNIQUE           -- 1:1 HEADS (department side is total)
);

CREATE TABLE professors (
  prof_id    INT PRIMARY KEY,
  name       VARCHAR(60) NOT NULL,
  salary     DECIMAL(10,2) CHECK (salary > 0),
  dept_id    INT NOT NULL REFERENCES departments(dept_id), -- N:1 WORKS_IN
  mentor_id  INT REFERENCES professors(prof_id)           -- recursive 1:N
);

ALTER TABLE departments
  ADD FOREIGN KEY (head_prof_id) REFERENCES professors(prof_id);

CREATE TABLE students (
  roll_no    INT PRIMARY KEY,
  first_name VARCHAR(40) NOT NULL,    -- composite name split
  last_name  VARCHAR(40),
  dob        DATE NOT NULL,           -- age is derived, not stored
  dept_id    INT NOT NULL REFERENCES departments(dept_id) -- N:1, total
);

CREATE TABLE student_phones (          -- multi-valued attribute
  roll_no    INT REFERENCES students(roll_no) ON DELETE CASCADE,
  phone      VARCHAR(15),
  PRIMARY KEY (roll_no, phone)
);

CREATE TABLE courses (
  course_code VARCHAR(10) PRIMARY KEY,
  title       VARCHAR(100) NOT NULL,
  credits     INT CHECK (credits BETWEEN 1 AND 6),
  dept_id     INT NOT NULL REFERENCES departments(dept_id) -- 1:N OFFERS
);

CREATE TABLE enrollments (            -- M:N junction table
  roll_no     INT REFERENCES students(roll_no),
  course_code VARCHAR(10) REFERENCES courses(course_code),
  semester    VARCHAR(10),
  grade       CHAR(2),
  PRIMARY KEY (roll_no, course_code, semester)
);

CREATE TABLE sections (               -- weak entity
  course_code VARCHAR(10) REFERENCES courses(course_code) ON DELETE CASCADE,
  section_no  INT,
  room        VARCHAR(20),
  PRIMARY KEY (course_code, section_no)
);

```

Notes on choices:

- `head_prof_id` is placed in `departments` (the total side of HEADS) with `UNIQUE`, so one professor cannot head two departments.

- The `departments` and `professors` tables reference each other, so one foreign key is added with `ALTER TABLE` after both exist. When inserting, create the department with `head_prof_id` `NULL`, insert professors, then update the head. (If the head is mandatory, you would enforce it with a deferred constraint or in application logic.)
- The `enrollments` primary key includes `semester` because a student may repeat a course in a later semester. Without it, the second attempt would be rejected. Always ask: "can this pair occur more than once?"
- Total participation of `STUDENT` in `BELONGS_TO` becomes `NOT NULL` on `dept_id`.

Table count check: 7 tables (`departments`, `professors`, `students`, `student_phones`, `courses`, `enrollments`, `sections`) from 5 entity sets, 1 multi-valued attribute and 1 M:N relationship. The 1:1, 1:N and recursive relationships added only columns.

2.7.3 Mapping specialization: three options

For `EMPLOYEE` specialised into `FULL_TIME` and `CONTRACTOR`:

Option	Tables	Pros	Cons	ORM name
A. Single table	<code>employees(emp_id, name, type, salary, pf_no, hourly_rate, agency)</code>	No joins, simple queries	Many <code>NULL</code> columns; weak constraints (salary cannot be <code>NOT NULL</code>)	Single table inheritance
B. Table per subclass + superclass	<code>employees(emp_id, name)</code> , <code>full_time(emp_id FK PK, salary, pf_no)</code> , <code>contractors(emp_id FK PK, hourly_rate, agency)</code>	No <code>NULL</code> s, clean constraints, works for overlapping	Joins needed to get full data	Joined / class table inheritance
C. Table per concrete subclass only	<code>full_time(emp_id, name, salary, pf_no)</code> , <code>contractors(emp_id, name, hourly_rate, agency)</code>	No joins for subclass queries	Common columns duplicated; "all employees" needs <code>UNION</code> ; only works for total, disjoint	Concrete table inheritance

DO YOU KNOW?

These three options map directly to JPA/Hibernate's inheritance strategies: `SINGLE_TABLE`, `JOINED` and `TABLE_PER_CLASS`. In JPA, `SINGLE_TABLE` is the default strategy. Django supports a similar idea with multi-table inheritance and abstract base classes.

2.7.4 Mapping aggregation

For "MANAGER monitors (EMPLOYEE works on PROJECT)": first map `WORKS_ON` (M:N) to `works_on(emp_id, project_id, PRIMARY KEY(emp_id, project_id))`. Then `MONITORS` (say each assignment has one monitoring manager) becomes a column `manager_id` in `works_on`, or, if M:N, a

table `monitors(emp_id, project_id, manager_id)` with a composite foreign key `(emp_id, project_id)` REFERENCES `works_on` .

INTERVIEW CORNER

Q: How do you convert an M:N relationship into tables?

A: Create a junction table containing the primary keys of both entities as foreign keys, plus any relationship attributes. Its primary key is the combination of the two foreign keys (or a surrogate key with a UNIQUE constraint on the pair).

Q: For a 1:N relationship, why put the foreign key on the N side and not the 1 side?

A: Each N-side row relates to only one 1-side row, so a single column holds it. On the 1 side, one row would need many values, which would violate atomic values (first normal form) or need repeating columns.

Q: Design tables for "users follow users" like Instagram.

A: It is a recursive M:N relationship: `follows(follower_id, followee_id, created_at, PRIMARY KEY(follower_id, followee_id))` , both columns referencing `users(id)` , with a CHECK that they are not equal and an index on `followee_id` for "who follows me" queries.

CHECK YOUR PROGRESS 2.7

1. An ER diagram has 3 strong entities, 1 weak entity, 2 M:N relationships, 2 1:N relationships and 1 multi-valued attribute. How many tables at minimum?

Answer: 3 (strong) + 1 (weak) + 2 (M:N) + 1 (multi-valued) = 7. The 1:N relationships add foreign key columns only.

2. Where does the relationship attribute "grade" go for the M:N ENROLLS relationship?

Answer: In the junction table `enrollments` , because grade belongs to the student-course pair.

3. How is total participation of the N side in a 1:N relationship enforced in SQL?

Answer: By declaring the foreign key column on the N side as NOT NULL.

2.8 Common Modelling Mistakes at Work

- Comma-separated lists in a column:** `tags = 'veg,spicy,bestseller'` . You cannot index, join or validate these. Use a junction table (`item_tags`) or, if you must, a proper array/JSON type with care.
- Repeating columns:** `phone1, phone2, phone3` . The fourth phone breaks the design. It is a multi-valued attribute; give it a table.
- Wrong cardinality:** assuming one address per user, then a customer wants "Home" and "Office". Ask both direction questions, and ask the business how it may change next year.
- Using a natural attribute as the primary key when it can change:** phone number or email as PK. People change numbers. Use a surrogate key and put UNIQUE on email.
- Storing derived data without a plan:** `order_total` stored and then line items edited, so they disagree.

6. **Missing junction table attributes:** forgetting that the price at time of order belongs in `order_items`, not only in `products`. If the product price changes, old invoices must not change.
7. **God table:** one `data` table with columns `type`, `key`, `value` for everything (the EAV anti-pattern). Flexible, but constraints, types and queries become painful.
8. **No foreign keys "for speed":** orphan rows pile up (order items for deleted orders) and nobody notices until a report is wrong.
9. **Polymorphic foreign key:** `comments(entity_type, entity_id)` pointing to posts or photos. The database cannot enforce it. Prefer separate nullable foreign keys or separate tables.
10. **Ignoring time:** modelling "current department" only, then HR asks "which department was she in last March?" Consider history tables or `valid_from/valid_to` columns.

😄 FUNNY SIDE

Product manager: "A user will only ever have one address." Six months later: "Small change, users want to save Home, Office, Mummy's house and Gym." The `address` column, quietly: "I knew this day would come."

👛 AT WORK

In a code review you see a migration adding `assigned_user_ids TEXT` to a `tickets` table, storing "12,45,78". The right review comment: "This is an M:N between tickets and users. Please add a `ticket_assignees(ticket_id, user_id)` table with a composite primary key and foreign keys. We will need 'all tickets assigned to user 45', which this column cannot index." Spotting this early saves a painful data migration later.

💡 DO YOU KNOW?

Peter Chen's paper "The Entity-Relationship Model: Toward a Unified View of Data" was published in ACM Transactions on Database Systems in 1976, and it is among the most cited papers in computer science.

✅ CHECK YOUR PROGRESS 2.8

1. Why is storing "veg,spicy" in a single `tags` column a bad idea?

Answer: The values are not atomic, so you cannot index or join on individual tags, enforce valid tag values, or easily count items per tag. A junction table solves all of these.

2. Why should the unit price be stored in `order_items` even though `products` has a `price` column?

Answer: Product prices change over time. The order must keep the price at the time of purchase so invoices and refunds remain correct.

What Have We Learnt?

- Database design goes from requirements to conceptual (ER), logical (tables) and physical (indexes, storage) design.
- Hierarchical (tree), network (graph with pointers), relational (tables with keys), object and document (JSON) are the main data models.
- An entity is a distinguishable thing; an entity set is all entities of a type. Attributes may be simple/composite, single/multi-valued, stored/derived and key.
- Relationships associate entities and can have their own descriptive attributes.
- Degree is the number of participating entity sets: unary, binary, ternary.
- Cardinality (1:1, 1:N, M:N) is the maximum; participation (total or partial) is the minimum. Ask both directions.
- A weak entity has only a partial key; its key is owner key + partial key; it has total participation in an identifying relationship (double rectangle, double diamond).
- Specialization (top-down) and generalization (bottom-up) create ISA hierarchies with inheritance; they can be disjoint/overlapping and total/partial.
- Aggregation lets a relationship participate in another relationship.
- Mapping: entity → table; multi-valued → new table; 1:N → FK on N side; 1:1 → FK with UNIQUE on total side; M:N → junction table; weak → composite PK with owner key.
- Specialization maps to single table, joined tables, or table per concrete class, matching ORM inheritance strategies.
- Common mistakes: comma lists, repeating columns, wrong cardinality, mutable natural keys, missing foreign keys, EAV and polymorphic keys.

Key terms: data model, conceptual design, logical design, physical design, hierarchical model, network model, relational model, object-relational, document model, entity, entity set, attribute, domain, simple attribute, composite attribute, multi-valued attribute, derived attribute, key attribute, relationship, relationship set, descriptive attribute, degree, unary, binary, ternary, cardinality, participation, total participation, partial participation, (min,max) notation, crow's foot, weak entity, owner entity, partial key, discriminator, identifying relationship, EER, specialization, generalization, inheritance, disjoint, overlapping, aggregation, junction table.

Exercises

A. Multiple Choice Questions

1. Which data model uses parent-child trees?
 - a. Relational
 - b. Hierarchical
 - c. Document
 - d. Graph
2. A double oval represents a:
 - a. Derived attribute
 - b. Key attribute
 - c. Multi-valued attribute
 - d. Weak entity
3. "Age" computed from date of birth is a:
 - a. Composite attribute
 - b. Derived attribute
 - c. Key attribute
 - d. Multi-valued attribute
4. The relationship between Student and Course is usually:
 - a. 1:1
 - b. 1:N
 - c. N:1
 - d. M:N
5. Total participation is drawn as:
 - a. A dashed line
 - b. A double line
 - c. An arrow
 - d. A triangle
6. The key of a weak entity is:
 - a. Its partial key alone
 - b. The owner's key alone
 - c. Owner's key + partial key
 - d. Any attribute
7. Combining CAR and BIKE into VEHICLE is:
 - a. Specialization
 - b. Generalization
 - c. Aggregation
 - d. Normalization
8. For a 1:N relationship, the foreign key goes in:
 - a. The 1 side table
 - b. The N side table
 - c. A new table always
 - d. Both tables
9. The degree of "Employee manages Employee" is:
 - a. 1
 - b. 2
 - c. 3
 - d. 0
10. Which feature lets a relationship take part in another relationship?
 - a. Inheritance
 - b. Specialization
 - c. Aggregation
 - d. Weak entity

B. Fill in the Blanks

1. The ER model was proposed by _____ in 1976.
2. A weak entity is drawn as a _____ rectangle.
3. The attribute that distinguishes weak entities of the same owner is called the _____ key.
4. An M:N relationship is mapped to a separate _____ table.
5. In specialization, an entity that can belong to more than one subclass is called _____.
6. Cardinality specifies the _____ number of relationship instances; participation specifies the minimum.

C. True or False

1. A relationship can have its own attributes.
2. A weak entity can have partial participation in its identifying relationship.
3. A multi-valued attribute is stored as multiple columns in the same table.
4. Specialization is a top-down process.
5. A 1:1 relationship always needs a separate table.

D. Short Answer Questions

1. Differentiate between composite and multi-valued attributes with examples.
2. What is a weak entity? Give two examples.
3. Explain cardinality with one example of each type.
4. What is aggregation and when is it used?
5. How is a multi-valued attribute converted into a table?
6. State any three common data modelling mistakes and their fixes.

E. Long Answer Questions

1. Compare the hierarchical, network, relational and document data models.
2. Explain generalization, specialization and inheritance with constraints (disjoint, overlapping, total, partial) and the three ways to map them to tables.
3. Write the complete set of rules for converting an ER diagram into relational tables, with an example of each.
4. Draw an ER diagram for a hospital (patients, doctors, appointments, prescriptions, wards) and convert it to tables.

F. Design Challenge

Design the ER model and tables for a movie ticket booking app like BookMyShow: cities, cinemas, screens, seats, movies, shows (a movie on a screen at a time), users, bookings and booked seats. A seat must not be booked twice for the same show. Identify weak entities and state cardinalities.

Answer Key: Chapter 2

A. MCQ

1. (b) Hierarchical model is a tree.
2. (c) Double oval means multi-valued.
3. (b) Computed values are derived.
4. (d) Many students, many courses.
5. (b) Double line means total.
6. (c) Owner's key plus partial key.
7. (b) Bottom-up is generalization.
8. (b) FK on the many side.
9. (a) One entity set, so unary (degree 1).
10. (c) Aggregation.

B. Fill in the Blanks

1. Peter Chen
2. double
3. partial (discriminator)
4. junction (associative / bridge)
5. overlapping
6. maximum

C. True or False

1. True. For example grade in ENROLLS.
2. False. It must have total participation, since it cannot be identified without its owner.
3. False. It is stored in a separate table with the owner's key.
4. True. It starts from a general entity and defines subclasses.
5. False. A foreign key with UNIQUE in one table is enough; the tables can even be merged.

D. Short Answers

1. A composite attribute has sub-parts but one value per entity, like name (first, last). A multi-valued attribute has several values per entity, like phone numbers. Composite attributes become several columns; multi-valued ones need a separate table.
2. A weak entity cannot be uniquely identified by its own attributes and depends on an owner entity; its key is owner key plus partial key. Examples: PAYMENT of a LOAN, DEPENDENT of an EMPLOYEE, SECTION of a COURSE, ROOM of a HOTEL.
3. Cardinality is the maximum number of entities one entity can relate to. 1:1: person and passport. 1:N: department and employees. M:N: students and courses.
4. Aggregation treats a relationship together with its entities as a higher-level entity so it can participate in another relationship. It is used when, for example, a manager monitors a specific employee-project assignment, not the employee or project alone.
5. Create a new table with the owner's primary key (as a foreign key) and the attribute value. The primary key is the combination, for example `student_phones(roll_no, phone)`.
6. Comma-separated lists in a column (fix: junction table); repeating columns like phone1, phone2 (fix: separate table); mutable natural key such as phone as PK (fix: surrogate key with UNIQUE on

the natural attribute); missing foreign keys (fix: add FK constraints).

E. Long Answers

1. Model answer points: hierarchical stores trees with one parent per child, fast traversal, poor at M:N (IBM IMS); network stores graphs with pointer sets, handles M:N but navigational code (CODASYL, IDMS); relational stores tables linked by key values with declarative SQL and strong theory (PostgreSQL, MySQL); document stores flexible JSON documents, good for whole-object access and horizontal scaling, weaker joins (MongoDB); mention data independence and query flexibility as reasons relational won for business data.
2. Model answer points: specialization top-down, generalization bottom-up, ISA triangle; subclasses inherit attributes and relationships; disjoint vs overlapping (one or many subclasses); total vs partial (must or may belong to a subclass); mapping options: single table with type column (NULLs, simple), superclass plus subclass tables joined by shared PK (clean, needs joins, supports overlap), table per concrete class (no joins, duplication, only total disjoint); relate to JPA `SINGLE_TABLE`, `JOINED`, `TABLE_PER_CLASS`.
3. Model answer points: strong entity → table with PK; composite → component columns; multi-valued → separate table; derived → not stored; weak entity → table with owner FK and composite PK, ON DELETE CASCADE; 1:1 → FK with UNIQUE on total side or merge; 1:N → FK on N side; M:N → junction table with composite PK and relationship attributes; recursive → self FK or junction with two FKs to same table; n-ary → table with FKs to all; specialization → three options; aggregation → outer relationship references the inner relationship's table.
4. Model answer points: entities `PATIENT`(patient_id, name, dob, phones multi-valued), `DOCTOR`(doctor_id, name, specialization), `WARD`(ward_no, type, beds), `MEDICINE`(med_id, name); `APPOINTMENT` as M:N between `PATIENT` and `DOCTOR` with date_time (or as an entity); `PRESCRIPTION` as weak entity of `APPOINTMENT` or ternary `DOCTOR-PATIENT-MEDICINE` with dosage; `ADMITTED_IN` as N:1 `PATIENT` to `WARD` with admit_date; tables: patients, patient_phones, doctors, wards, medicines, appointments(appt_id, patient_id, doctor_id, slot, UNIQUE(doctor_id, slot)), prescription_items(appt_id, med_id, dosage, PK(appt_id, med_id)), admissions(patient_id, ward_no, admit_date, discharge_date).

F. Design Challenge: Model Solution

Entities: `CITY`, `CINEMA` (N:1 city), `SCREEN` (weak entity of `CINEMA`, partial key screen_no), `SEAT` (weak entity of `SCREEN`, partial key seat_label like "A5"), `MOVIE`, `SHOW` (a `SCREEN` shows a `MOVIE` at a start_time; N:1 to both), `USER`, `BOOKING` (N:1 `USER`, N:1 `SHOW`). `BOOKED_SEAT` is M:N between `BOOKING` and `SEAT`. Tables: cities(city_id PK, name), cinemas(cinema_id PK, name, city_id FK NOT NULL), screens(cinema_id FK, screen_no, PK(cinema_id, screen_no)), seats(cinema_id, screen_no, seat_label, seat_type, PK(cinema_id, screen_no, seat_label), FK(cinema_id, screen_no) → screens), movies(movie_id PK, title, duration_min, language), shows(show_id PK, movie_id FK, cinema_id, screen_no, start_time, FK(cinema_id, screen_no) → screens, UNIQUE(cinema_id, screen_no, start_time)), users(user_id PK, email UNIQUE, phone), bookings(booking_id PK, user_id FK, show_id FK, status, amount, created_at), booked_seats(booking_id FK, show_id, seat_label, price, PRIMARY KEY(show_id, seat_label)). The primary key (show_id, seat_label) on booked_seats makes the database itself reject double booking of the same seat for the same show, even when two users click at the same moment. (In practice the screen is known from the show; you may add a composite

foreign key to seats for full integrity.) Cardinalities: city 1:N cinema, cinema 1:N screen, screen 1:N seat, screen 1:N show, movie 1:N show, user 1:N booking, show 1:N booking, booking M:N seat.