

Databases

A-Level Computer Science

File-based storage and its limits

Before databases, programs stored data in **flat files** 平面文件 — usually one file per program. This is fine for small data but breaks down at scale.



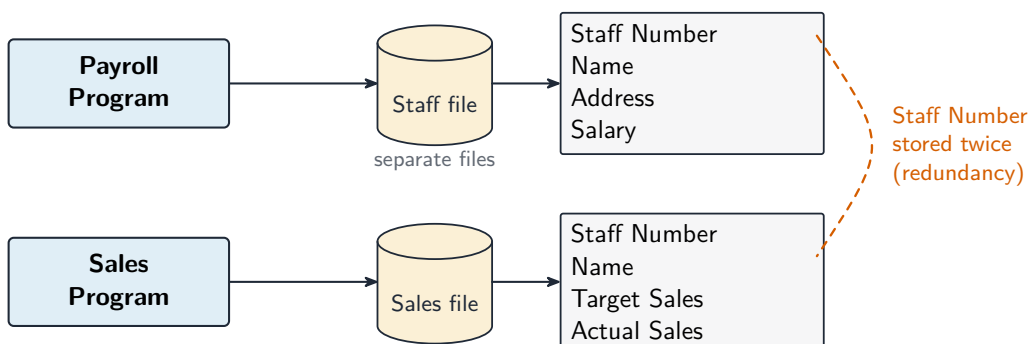
File-based storage keeps data in separate files, like papers in a filing cabinet — hard to search and easy to duplicate

Limitations

- **data redundancy** 数据冗余 — the same data (a customer's address) is held in several files.
- **data inconsistency** 数据不一致 — redundant copies updated separately get out of sync.
- **data dependence** — programs are tied to the file format; change the format and every program must be rewritten.
- hard to enforce **integrity** 完整性, hard to share safely, weak querying, and weak per-field security.

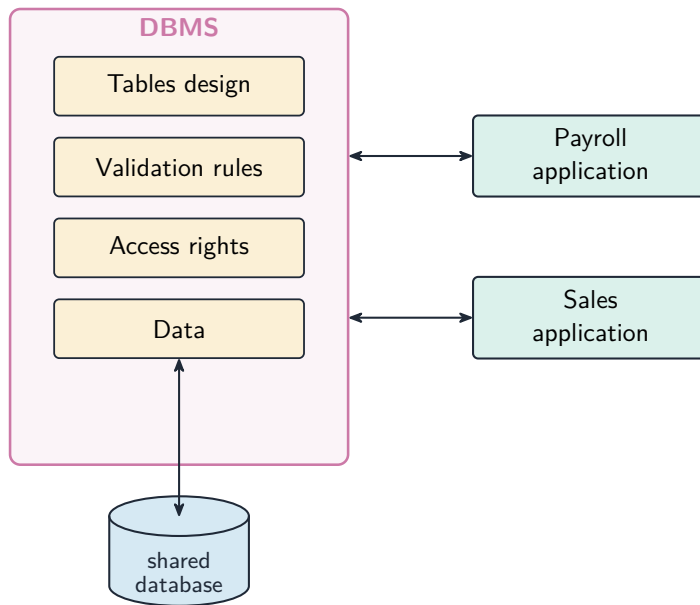


The files themselves are stored on devices such as hard disk drives



The file-based approach: each program keeps its own files

A **relational database** 关系数据库 fixes these by storing data in **tables** managed by one piece of software (the DBMS) that all programs use.



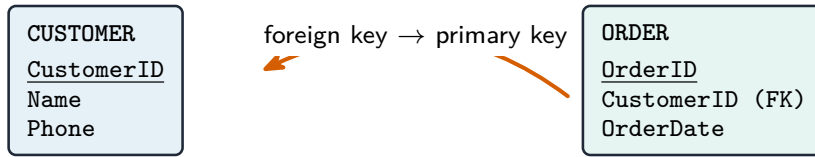
The database approach: one DBMS serves all the programs

Relational model — terms

- **table** 表 (relation) — a grid of rows and columns; one table per type of **entity** 实体 (e.g. CUSTOMER).
- **record** 记录 (row, also called a **tuple** 元组) — one row; one instance of the entity.
- **field** 字段 (column, also called an **attribute** 属性) — one column; one piece of information about each record.
- **primary key** 主键 — a field (or fields) that **uniquely identifies** each record; never null or duplicated.
- **foreign key** 外键 — a field whose value matches the primary key of another table, linking the two.
- **composite key** 复合键 — a primary key made of two or more fields together.
- **candidate key** 候选键 — any field(s) that could be the primary key.
- **secondary key** 次键 — a non-primary field that is indexed for fast searching.
- **indexing** 索引 — building an index on a field so look-ups and joins run faster.
- **referential integrity** 参照完整性 — every foreign-key value must match an existing primary key (no orphan records).

A table is written in shorthand with the primary key underlined and foreign keys noted:

```
CUSTOMER(CustomerID, Name, Phone)
ORDER(OrderID, CustomerID, OrderDate)    -- CustomerID is FK → CUSTOMER
```

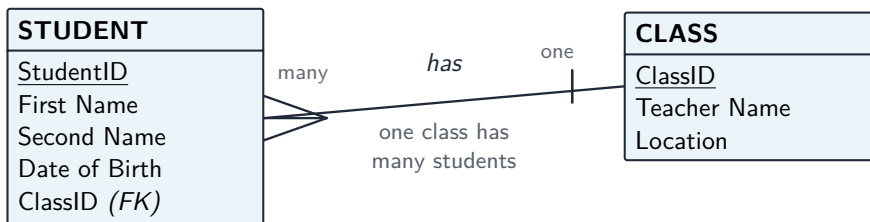


A foreign key links two tables: ORDER.CustomerID matches the primary key CUSTOMER.CustomerID

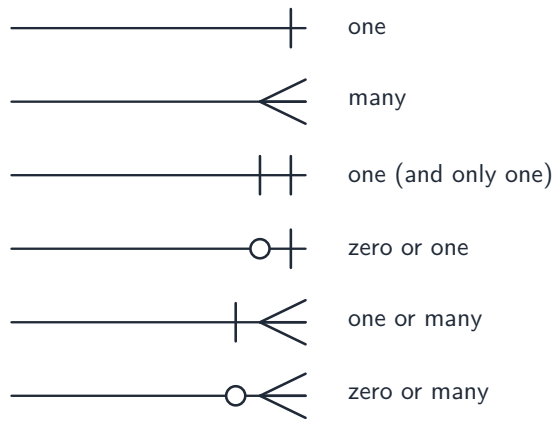
Entity-relationship (E-R) diagrams

An **entity-relationship diagram** 实体关系图 shows the structure: each entity is a rectangle, each relationship a line, with the **cardinality** 基数 marked at each end:

- **one-to-one** (1:1).
- **one-to-many** 一对多 (1:M) — each Customer has many Orders; each Order has one Customer.
- **many-to-many** (M:N) — Students take many Courses, and Courses have many Students.



An E-R diagram: one class has many students



Crow's-foot symbols for the cardinality of a relationship

A many-to-many relationship cannot be stored directly. Break it into two one-to-many relationships through a **link table** 连接表 holding the two foreign keys:

```
ENROLMENT(StudentID, CourseID, EnrolmentDate)
```



a link table turns many-to-many into two one-to-many relationships

A link table resolves a many-to-many relationship into two one-to-many relationships

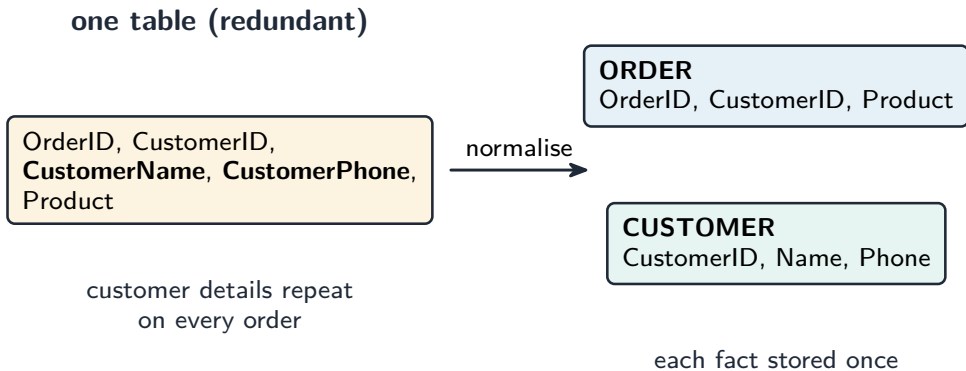
Normalisation

Normalisation 规范化 organises tables to **cut redundancy and inconsistency**, going through **normal forms** 范式 in order.

- **First normal form (1NF)** — every field holds a single (**atomic** 原子) value, with no repeating groups, and a primary key.
- **Second normal form (2NF)** — in 1NF, and every non-key field depends on the **whole** primary key (only matters for a composite key).
- **Third normal form (3NF)** — in 2NF, and every non-key field depends **only** on the primary key, not on another non-key field (no **transitive dependency** 传递依赖).

A 3NF design stores each fact once, so insert/update/delete anomalies disappear. The trade-off is more tables and more joins. Aim for 3NF.

To produce a 3NF design: find the entities and their attributes; choose a primary key for each; split repeating/non-atomic fields (1NF); split fields depending on part of a composite key (2NF); split fields depending transitively on the key (3NF); add foreign keys for the relationships.



Normalisation removes redundancy by splitting repeated data into its own table

Worked example. The table ORDER(OrderID, CustomerID, CustomerName, ProductID, Quantity) has the composite primary key (OrderID, ProductID). Normalise it to 3NF. Test each non-key field against the key. Quantity depends on **both** OrderID and ProductID, which is fine. But CustomerID depends on **OrderID alone** - only *part* of the composite key. That is a **partial dependency**, so the table is not in **2NF**. Split it into ORDER_LINE(OrderID, ProductID, Quantity) and ORDER(OrderID, CustomerID, CustomerName). Now test 3NF: in that new ORDER table, CustomerName depends on CustomerID, which is **not** the key - a **transitive dependency**. Split again: ORDER(OrderID, CustomerID) and CUSTOMER(CustomerID, CustomerName). Name the **dependency** that breaks each form (partial breaks 2NF, transitive breaks 3NF); "it has repeated data" describes the symptom and earns nothing.

Database Management System (DBMS)

A **DBMS** 数据库管理系统 manages the database centrally. Features that fix the file-based limits:

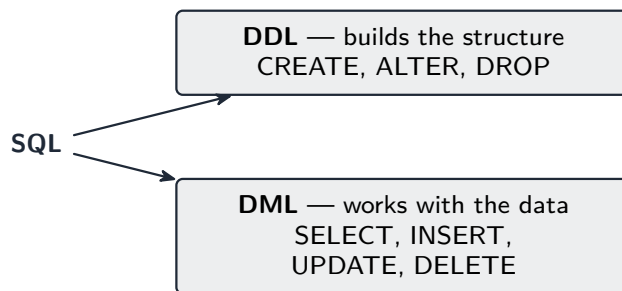
- **data dictionary** 数据字典 — a description of every table, field, type and key; programs query it instead of hard-coding the structure.
- **redundancy/consistency control** — each fact stored once.

- **concurrent access** 并发访问 control — locks and transactions let many users work at once.
- **backup** 备份 and recovery; security and per-user permissions.
- **integrity rules** — keys, unique and range constraints, enforced centrally.
- **transactions** 事务 — a group of operations that all succeed or all fail.
- **views** 视图 — virtual tables that show each user “their” slice of the data.
- **data management** 数据管理 and **data modelling** 数据建模 — control how data is stored and define its structure as a **logical schema** 逻辑模式 (the logical design, independent of physical storage).
- **data integrity** 数据完整性 and **data security** 数据安全 — enforce correctness and control access centrally.
- a **query processor** 查询处理器 runs queries; a **developer interface** 开发者接口 gives tools and APIs for building applications.

Its tools include a data-dictionary editor, a query builder, a forms builder, a report generator, user management, and an SQL editor.

DDL and DML

SQL 结构化查询语言 (Structured Query Language) has two halves:



DDL builds the database structure; DML works with the data

- **Data Definition Language** 数据定义语言 (DDL) — creates or changes the **structure** (tables, keys, constraints).
- **Data Manipulation Language** 数据操纵语言 (DML) — works with the **data** (insert, update, delete, **query** 查询).

DDL basics

```
CREATE TABLE CUSTOMER (  
  CustomerID INTEGER PRIMARY KEY,  
  Name VARCHAR(50) NOT NULL,  
  Phone VARCHAR(20)  
);
```

Add a foreign key:

```
CREATE TABLE ORDER (  
  OrderID INTEGER PRIMARY KEY,  
  CustomerID INTEGER,  
  OrderDate DATE,  
  FOREIGN KEY (CustomerID) REFERENCES CUSTOMER(CustomerID)  
);
```

Modify and drop:

```
ALTER TABLE CUSTOMER ADD Email VARCHAR(100);  
DROP TABLE CUSTOMER;
```

Common types: INTEGER, REAL, VARCHAR(n), CHAR(n) (also CHARACTER(n)), DATE, TIME, BOOLEAN, DECIMAL(p, s).

DML basics

Query with SELECT:



A SELECT query returns only the rows that match its condition

```
SELECT Name, Phone  
FROM CUSTOMER  
WHERE City = 'London'  
ORDER BY Name ASC;
```

SELECT lists fields, FROM names the table, WHERE filters rows, ORDER BY sorts.

A **join** 连接 combines two tables using a foreign-key relationship:

```
SELECT C.Name, O.OrderDate
FROM CUSTOMER C INNER JOIN ORDER O
  ON C.CustomerID = O.CustomerID
WHERE O.OrderDate >= '2024-01-01';
```

Aggregate functions 聚合函数 (COUNT, SUM, AVG, MIN, MAX) are often used with GROUP BY:

```
SELECT CustomerID, COUNT(*) AS NumOrders
FROM ORDER
GROUP BY CustomerID;
```

Insert, update, delete:

```
INSERT INTO CUSTOMER (CustomerID, Name, Phone)
VALUES (101, 'Ada Lovelace', '020-1234-5678');

UPDATE CUSTOMER SET Phone = '020-9999-0000' WHERE CustomerID = 101;

DELETE FROM CUSTOMER WHERE CustomerID = 101;
```

Always put a WHERE clause on UPDATE and DELETE, or the change hits **every** row.

Tips for exam SQL

- use the exact table and field names from the question.
- quote strings with single quotes ('Smith'); don't quote numbers.
- comparisons: =, <, >, <=, >=, <>.
- LIKE 'A%' matches anything starting with A (% = any string, _ = one character); IN (1,2,3); BETWEEN 10 AND 20.
- combine conditions with AND / OR / NOT, and end each statement with a semicolon.

Exam tips

- Define the terms exactly: entity, attribute, **primary key**, **foreign key**, and the relationship types (1:1, 1:many, many:many).
- Give a reason at each normal form: **1NF** (no repeating groups), **2NF** (no partial dependency), **3NF** (no non-key dependency).
- Explain what a **DBMS** provides (data independence, security, integrity, concurrent access).
- Distinguish **DDL** (define the structure) from **DML** (query and change the data).