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

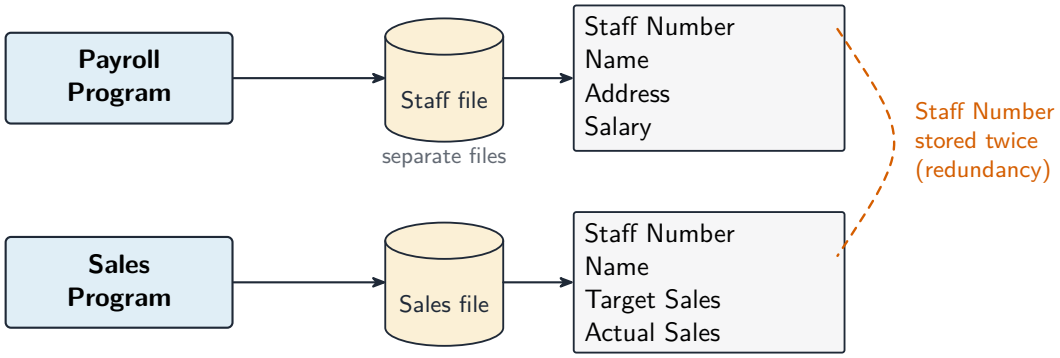
Image: Federal Bureau of Investigation, Public domain (commons.wikimedia.org)

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.

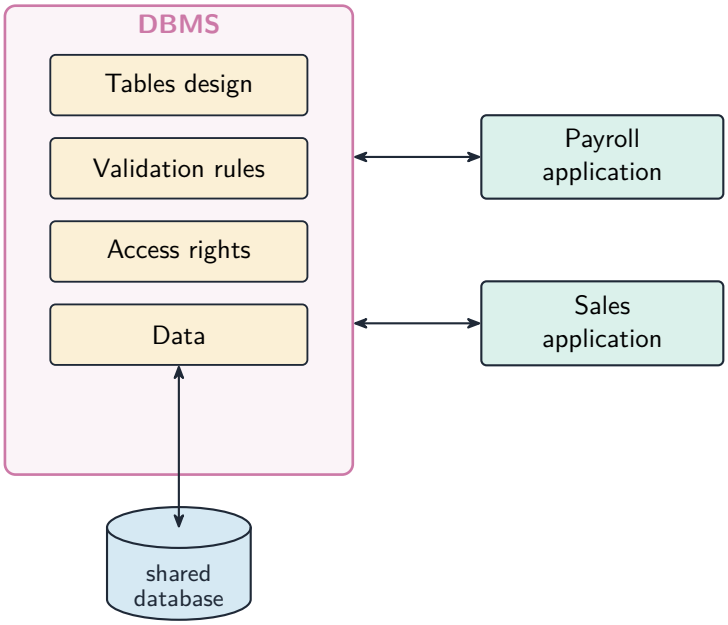
The files themselves are stored on devices such as hard disk drives

Image: RubinObs/NOIRLab/SLAC/NSF/DOE/AURA/J. Pinto, CC BY 4.0 (commons.wikimedia.org)



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) —one row; one instance of the entity.
- **field** 字段 (column) —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.
- **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:

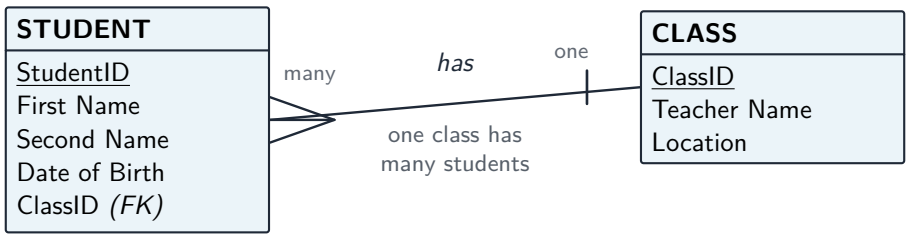
```
CUSTOMER(CustomerID, Name, Phone)
ORDER(OrderID, CustomerID, OrderDate)   -- CustomerID is FK → CUSTOMER
```

Entity-relationship (E-R) diagrams

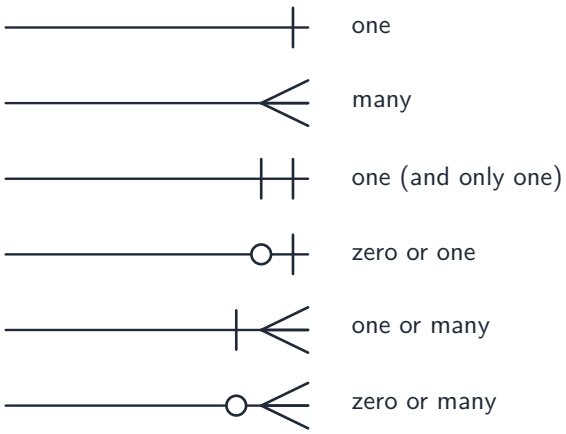
An **entity-relationship diagram** 实体关系图 shows the structure: each entity is a rectangle, each relationship a line, with the **cardinality** 基数 marked at each end:

- **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)
```

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.

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.

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:

- **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:

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

Common types: `INTEGER`, `VARCHAR(n)`, `CHAR(n)`, `DATE`, `BOOLEAN`, `DECIMAL(p, s)`.

DML basics

Query with `SELECT`:

```sql
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:

```sql
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`:

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

Insert, update, delete:

```sql
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.