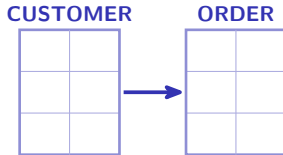


Databases

AS Computer Science ■ Topic 8

A-Level Computer Science



What we will cover

- 1 Files vs the database approach
- 2 The relational model
- 3 E-R diagrams
- 4 Normalisation
- 5 DBMS & SQL
- 6 Recap





Why not just use files?

Separate files for each program duplicate data and go out of step.

duplication

the same customer stored in sales, accounts and support

inconsistency

one file is updated; the others are not

no sharing

each program owns its files – no single source of truth

hard queries

“all orders for this customer” means writing a new program

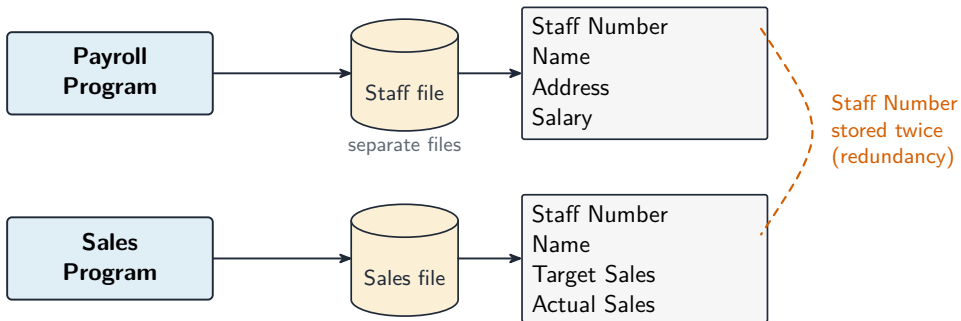
File-based storage and its limits

Before databases, programs stored data in **flat files** – usually one file per program.

Fine for small data; at scale it gives **redundancy** (the same fact in several files), **inconsistency** (they disagree), **dependence** on the program's own format, and no shared security.

Before databases: one file per program

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

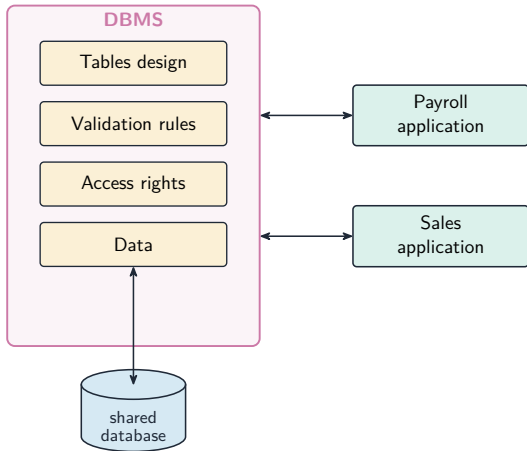


The files sit on the storage devices



Flat files on a disk: each program owns its own, which is exactly where the trouble starts.

The database approach

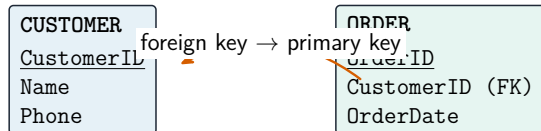


A **relational database** stores data in **tables**, managed by one **DBMS** that all programs share. Each fact is stored **once**.



The relational model

- a **table** of **records** (rows, also **tuples**)
- and **fields** (columns, also **attributes**)
- **primary key** – identifies a record uniquely
- **foreign key** – matches a primary key in another table



Referential integrity : no foreign key may point at a record that is not there.

The other kinds of key

composite key

a primary key made of two or more fields together, because no single field is unique

candidate key

any field, or set of fields, that *could* have been chosen as the primary key

secondary key

a non-primary field kept **indexed**, so searches and joins on it run fast

Indexing costs space and slows writes, so you index the fields you actually search on – not every field.



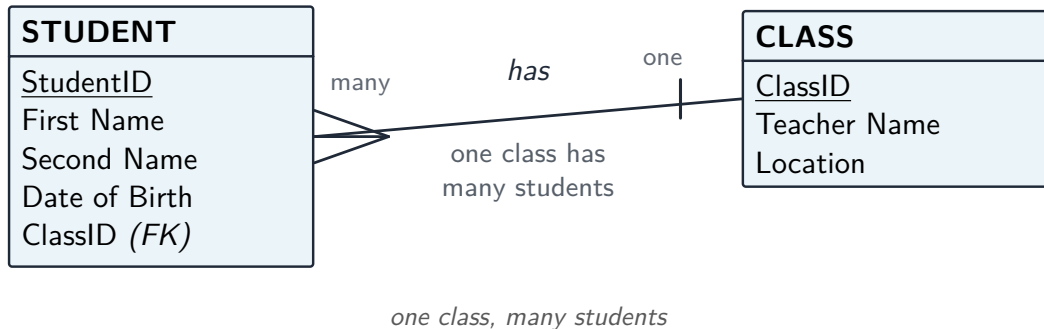
Entity-relationship diagrams

Cardinality at each end:

- one-to-one (1:1)
- **one-to-many** (1:M)
- **many-to-many** (M:N)

M:N needs a **link table** of two foreign keys.

Entity-relationship diagrams, in detail



An E-R diagram: one class has many students

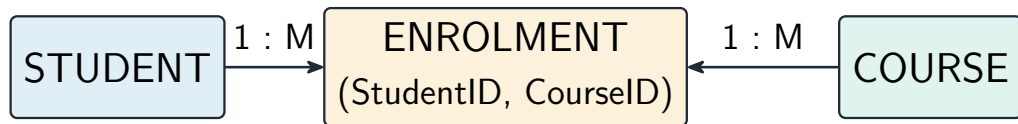
Crow's-foot notation

Read the symbol at each END of the line: it gives that side's **cardinality** .

	one
	many
	one (and only one)
	zero or one
	one or many
	zero or many

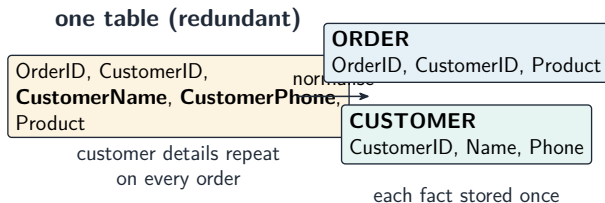
A link table breaks up M:N

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



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

Normalisation



Normalisation cuts redundancy through **normal forms** :

- **1NF**: atomic fields, a key
- **2NF**: depend on the **whole** key
- **3NF**: no **transitive dependency**

Find the entities and attributes, pick a primary key for each, split repeating fields (1NF), split fields depending on part of the key (2NF), then on a non-key field (3NF).

Worked example: to third normal form

Not 3NF: TeacherName depends on ClassID, not on the key –

StudentID	Name	ClassID	TeacherName
S01	Li Wei	C1	Mr Chen
S02	Anna	C1	Mr Chen
S03	Ben	C2	Ms Roy

split off the transitive fact



StudentID	Name	ClassID
S01	Li Wei	C1
S02	Anna	C1
S03	Ben	C2

FK



ClassID	TeacherName
C1	Mr Chen
C2	Ms Roy

Now “Mr Chen” is stored **once** – renaming a teacher touches one row.



The DBMS

A **DBMS** manages the database centrally – **data management** and **data modelling** define the **logical schema**, and it also handles **backup** and recovery, per-user security, **transactions** that wholly succeed or wholly fail, a **query processor** to run queries, and a **developer interface** of tools and APIs:

data dictionary

concurrent access

transactions

views

backup & security

query processor

What a DBMS provides, and the problems it removes

the tools

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

data redundancy

file-based storage repeats the same fact in several files; a database stores it once

data inconsistency

which is what happens when one copy is updated and another is not

data integrity

constraints and keys enforced by the DBMS itself, not by each program

data security

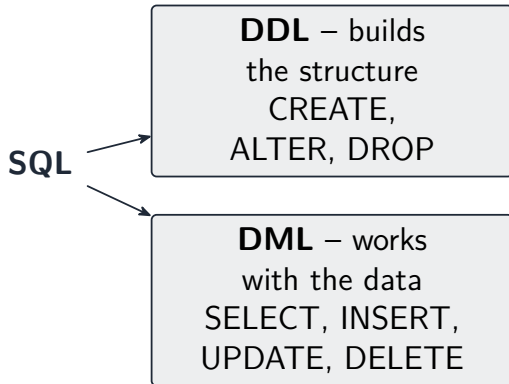
one place to set who may read and who may write

data independence

programs are written against the logical view, so the storage can change without rewriting them

Redundancy causes inconsistency, and inconsistency destroys integrity. The three arguments are one chain, and the exam expects it stated as one.

SQL – DDL and DML



DDL

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

- **DDL** : structure
- **DML** : the data

SQL – querying data

SELECT

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

A **join** combines tables on a foreign key; **aggregate functions** (COUNT, SUM) pair with GROUP BY.

```
SELECT name, grade  
FROM students  
WHERE grade > 80;
```

result

name	grade
Ann	88
Cara	91

Always put a WHERE on UPDATE/DELETE, or it hits every row.

SQL's two halves, and the exam's SQL rules

DDL

Data Definition Language – creates or changes the **structure**: CREATE TABLE, ALTER TABLE, keys and constraints

DML

Data Manipulation Language – works on the **data**: SELECT, INSERT, UPDATE, DELETE

the types

INTEGER, REAL, VARCHAR(n), CHAR(n), DATE, TIME, BOOLEAN, DECIMAL(p,s)

exact names

use the **table and field names from the question**, spelled as given

quoting

single quotes round strings ('Smith'); **do not quote numbers**

the operators

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 every statement with a semicolon**. Marks are lost on those details, not on the logic.

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).



Topic 8 – key takeaways

1

Why databases

remove redundancy &
inconsistency

2

Keys

primary identifies; foreign links
tables

3

Normalisation

1NF $\rightarrow$ 2NF $\rightarrow$ 3NF; each fact
once

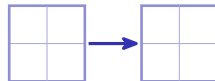
4

SQL

DDL builds structure; DML
queries data

Check yourself

- 1 What links one table to another?
- 2 How do you store a many-to-many relationship?
- 3 Which SQL keyword filters rows?
- 4 What does 3NF remove?



Answers 1. a foreign key 2. a link table 3. WHERE 4. transitive dependencies