

Databases

A-Level Computer Science 9618 · Topic 8 revision notes

Everything from the 6 lessons of this topic on a few pages: the key idea of each lesson, the diagram to remember, the points to learn and the traps examiners set. Read it, cover it, test yourself.

- | | |
|--|--|
| 1. <u>Relational databases and keys</u> | 4. <u>Database management systems</u> |
| 2. <u>Entity-relationship diagrams</u> | 5. <u>SQL data definition language</u> |
| 3. <u>Normalisation to third normal form</u> | 6. <u>SQL data manipulation language</u> |

thelucidstem.com

1 Relational databases and keys

Syllabus 8.1

In a **file-based approach** each application keeps its own files, which leads to **data redundancy** (the same data stored many times), **data inconsistency** (copies that disagree) and **data dependency** (programs tied to the file structure). A **relational database** stores data about each **entity** in its own **table** (relation) of **records** (tuples) and **fields** (attributes). Each table has a **primary key** that uniquely identifies every record, and tables are linked by **foreign keys**. **Referential integrity** means every foreign key value must match an existing primary key value.

a foreign key links two tables

MEMBER		LOAN		
<u>MemberID</u>	FirstName	<u>LoanID</u>	BookID	MemberID
M01	Aisha	L01	B101	M01
M02	Ben	L02	B103	M02
M03	Chloe	L03	B102	M01
M04	Dev	L04	B105	M03

primary key

foreign key

each value must match a primary key

Fig. 1: *MemberID* is the primary key of *MEMBER* and a foreign key in *LOAN*

Worked example: Redundancy leads to inconsistency

A student moves house. The office updates her address in the admissions file, but the library file still holds the old one. Name and explain the two problems shown.

1. The address is stored in two files: this is **data redundancy**.
2. Only one copy was updated, so the two files now give different addresses: this is **data inconsistency**.
3. In a relational database the address is stored once, in the *STUDENT* table, so there is only one copy to update.

Answer: Data redundancy causes data inconsistency

Name it, then explain it.

- ✓ **File-based problems:** redundancy, inconsistency, data dependency, poor integrity, security and sharing.
- ✓ **Keys:** primary (chosen, unique), candidate (could be primary), secondary (candidate not chosen, indexed), foreign (links to another table's primary key).
- ✓ **Referential integrity:** every foreign key value matches an existing primary key value.
- ✓ **Use the scenario:** name the actual tables and fields in your answer.

Watch out!

Redundancy and inconsistency are not the same thing. Redundancy is the duplication; inconsistency is the result when duplicated copies are not all updated. Name the right one for the marks.

Referential integrity is about foreign keys matching primary keys. It is not the same as validation (checking a value is sensible) or the uniqueness of a primary key.

2 Entity-relationship diagrams

Syllabus 8.1

An **entity-relationship (E-R) diagram** documents a database design. Each **entity** is a box, and each **relationship** is a line. The degree of a relationship is **one-to-one** (1:1), **one-to-many** (1:M) or **many-to-many** (M:M). In crow's foot notation the forked end marks the **many** side. A 1:M relationship is implemented by putting the primary key of the "one" table into the "many" table as a **foreign key**. A M:M relationship cannot be implemented directly: it is resolved with a **link entity** that has a 1:M relationship with each original entity.

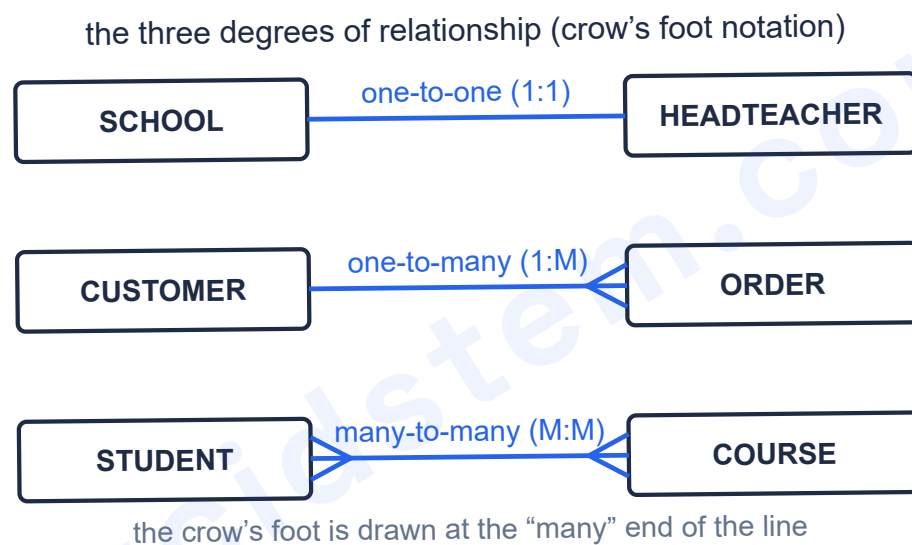


Fig. 1: one-to-one, one-to-many and many-to-many relationships

Worked example: Find the degree

In a hospital, each patient is placed in one ward and a ward holds many patients. State the relationship between WARD and PATIENT.

1. One ward has how many patients? Many.
2. One patient is in how many wards? One.
3. So the relationship is one-to-many, with the crow's foot at the PATIENT end.

Answer: WARD to PATIENT is one-to-many (1:M)

Ask it both ways.

- ✓ **Degree:** one A has how many B, and one B has how many A.
- ✓ **Notation:** crow's foot at every "many" end, plain line at every "one" end.
- ✓ **Foreign key:** the "one" side's primary key goes into the "many" side.
- ✓ **M:M:** add a link entity with a composite key made of both foreign keys.

Watch out!

Always ask the question in both directions. "A customer places many orders" on its own does not tell you whether the relationship is 1:M or M:M.

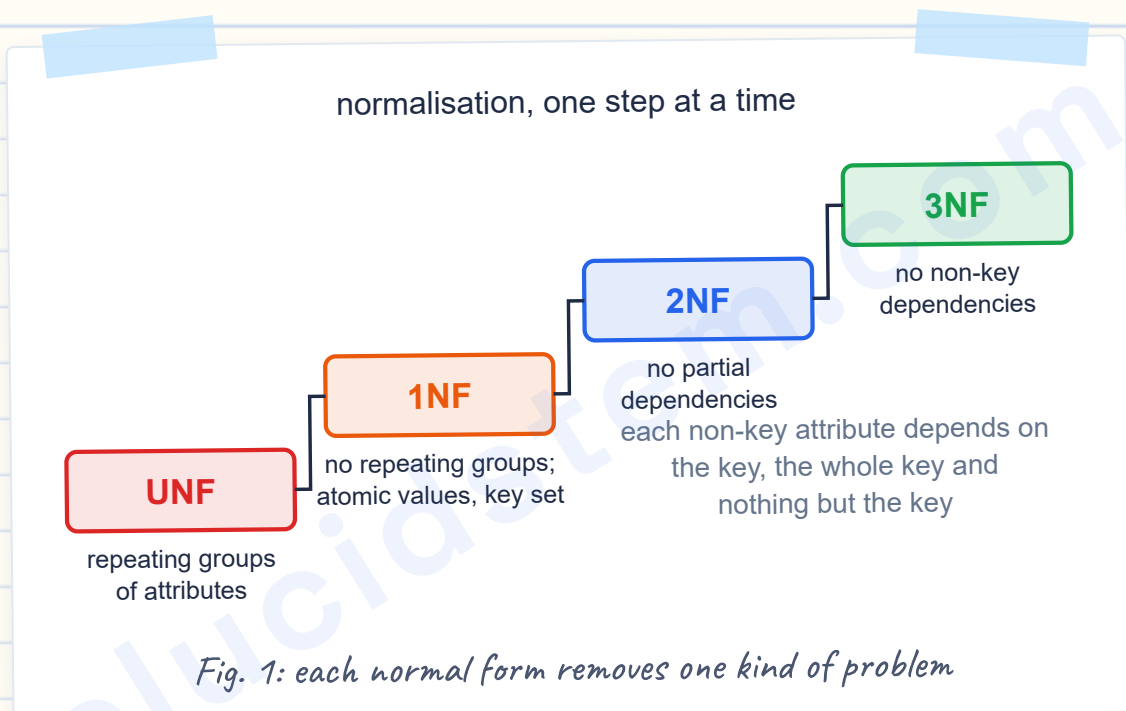
The crow's feet of the new relationships both point at the link entity. Drawing them pointing at ACTOR or FILM loses the marks for the resolved diagram.

thelucidstem.com

3 Normalisation to third normal form

Syllabus 8.1

Normalisation organises data into tables so that each fact is stored once, removing redundancy and the inconsistencies it causes. **First normal form (1NF)**: no repeating groups of attributes, every attribute holds a single (atomic) value, and a primary key identifies each tuple. **Second normal form (2NF)**: in 1NF, and no **partial dependencies**: every non-key attribute depends on the **whole** of a composite primary key. **Third normal form (3NF)**: in 2NF, and no **non-key dependencies**: no non-key attribute depends on another non-key attribute.



Worked example: Choosing the 1NF key

Why is StudentID on its own not a suitable primary key for the 1NF table above?

1. After removing the repeating group, S1 appears in two rows and S3 appears in two rows.
2. So StudentID no longer identifies a single tuple.
3. StudentID and CourseCode together are unique for every row, so they form a composite primary key (both underlined).

Answer: The primary key must be the composite StudentID + CourseCode

Three checks, in order.

- ✓ **1NF:** remove repeating groups; every field atomic; choose the (often composite) primary key.
- ✓ **2NF:** move attributes that depend on only part of a composite key into their own table.
- ✓ **3NF:** move attributes that depend on another non-key attribute into their own table, leaving a foreign key.
- ✓ **Write tables as** NAME(Key, Attribute, ...) and say which attributes are foreign keys.

Watch out!

1NF is not finished just because each cell holds one value. You must also state the new primary key, and in a table like this it is a composite key.

"It is not in 3NF because it has repeated data" earns little. Name the dependency: say which non-key attribute depends on which other attribute.

4 Database management systems

Syllabus 8.2

A **database management system (DBMS)** is the software that sits between users and the data. It provides **data management**, including a **data dictionary** (data about the data: tables, fields, data types, keys, validation rules and relationships), **data modelling** tools, a **logical schema** (the overall structure of the database, independent of how it is stored), **data integrity** (validation and referential integrity) and **data security** (backup procedures and **access rights** for individuals and groups). Developers build the database through the **developer interface**, and every query is checked and run by the **query processor**.

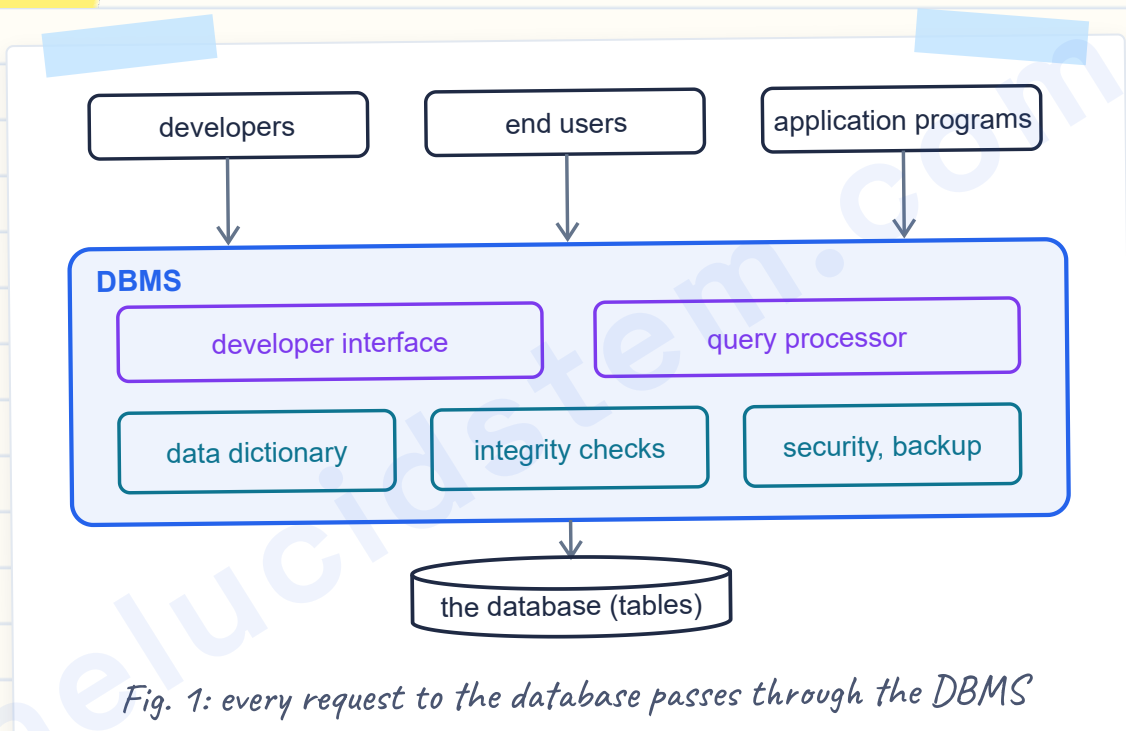


Fig. 1: every request to the database passes through the DBMS

Worked example: A data dictionary entry

Give the data dictionary entries the DBMS might hold for the field MemberID in the library database.

1. Table: MEMBER. Field name: MemberID.
2. Data type: VARCHAR(4); format: the letter M followed by two digits.
3. Key: primary key of MEMBER; also used as a foreign key in LOAN, so referential integrity applies.
4. Validation: presence check, format check; must be unique.

Answer: Name, table, data type and size, key status, relationships and validation rules

Feature, then purpose.

- ✓ **Data dictionary:** metadata, not the data.
- ✓ **Integrity:** validation, unique keys, referential integrity. **Security:** passwords, access rights, backups.
- ✓ **Access rights:** per user or group: none, read-only, read and write.
- ✓ **Developer interface** builds; **query processor** runs queries.

Watch out!

The data dictionary holds data about the data (metadata), not the data itself. "It stores all the records" is wrong.

Do not confuse the two tools. The developer interface is used to build the database and its forms, queries and reports; the query processor runs queries.

thelucidstem.com

5 SQL data definition language

Syllabus 8.3

The DBMS carries out all creation and modification of the database **structure** using its **data definition language (DDL)**, and all queries and maintenance of the **data** using its **data manipulation language (DML)**. The industry standard for both is **SQL**. The DDL statements you need are **CREATE DATABASE**, **CREATE TABLE** (each field with a data type: CHARACTER, VARCHAR(n), BOOLEAN, INTEGER, REAL, DATE, TIME), **ALTER TABLE**, **PRIMARY KEY (field)** and **FOREIGN KEY (field) REFERENCES Table(Field)**.

anatomy of a CREATE TABLE statement

```
CREATE TABLE Club( ----- the new table's name
ClubID VARCHAR(3),
ClubName VARCHAR(25), ----- one line per field:
                           name, then data type
StartTime TIME,
Indoor BOOLEAN,
PRIMARY KEY (ClubID) ----- the key, after the fields
);
```

commas separate the lines; the statement ends with a semicolon

Fig. 1: CREATE TABLE names the table, defines each field and declares the primary key

Worked example: Choosing data types

Choose a suitable SQL data type for each field of a CLUB table: ClubID (codes such as C01), ClubName, StartTime, Indoor (whether the club meets indoors), MaxMembers.

1. ClubID: a short code of 3 characters, VARCHAR(3).
2. ClubName: text of varying length, VARCHAR(25).
3. StartTime: a time of day, TIME. Indoor: yes or no, BOOLEAN.
4. MaxMembers: a whole number, INTEGER.

Answer: VARCHAR(3), VARCHAR(25), TIME, BOOLEAN, INTEGER

Name, type, comma.

- ✓ **DDL** changes structure (CREATE, ALTER); **DML** works with data (SELECT, INSERT, UPDATE, DELETE).
- ✓ **Types:** CHARACTER, VARCHAR(*n*), BOOLEAN, INTEGER, REAL, DATE, TIME. IDs and phone numbers are text.
- ✓ **Keys:** PRIMARY KEY (field); FOREIGN KEY (field) REFERENCES Table(Field).
- ✓ **Check:** brackets matched, commas between lines, a semicolon at the end.

Watch out!

An ID such as M01 or a phone number such as 07700 900123 is text, not INTEGER: it can have letters or a leading zero, and you never do arithmetic with it.

A foreign key field must have the same data type as the primary key it references. MemberID VARCHAR(4) in MEMBER needs MemberID VARCHAR(4) in LOAN, not INTEGER.

ALTER TABLE changes the structure. To change the values stored in a field you need the DML statement UPDATE, covered in the next lesson.

6 SQL data manipulation language

Syllabus 8.3

A query uses **SELECT** fields **FROM** table, an optional **WHERE** condition, **GROUP BY** to make one output row per group, and **ORDER BY** to sort, in that order. The aggregate functions **SUM**, **COUNT** and **AVG** work on a whole column, or on each group when **GROUP BY** is used. **INNER JOIN ... ON** combines two tables, outputting a row for each pair of records whose linking fields match. Data is maintained with **INSERT INTO**, **UPDATE ... SET ... WHERE** and **DELETE FROM ... WHERE**.

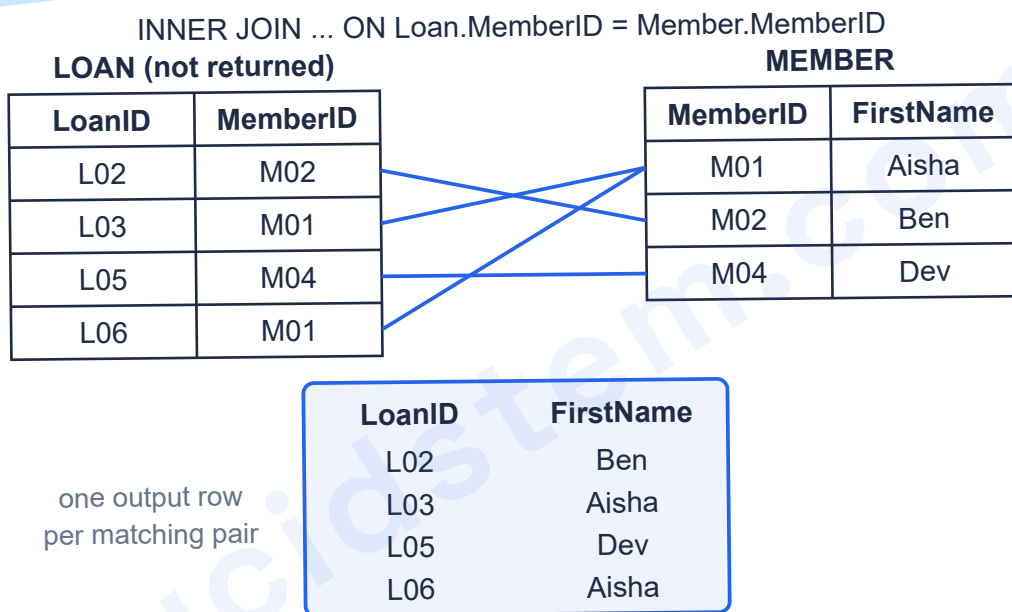


Fig. 1: **INNER JOIN** pairs each loan with the member whose *MemberID* matches

Worked example: **AVG** per group

Give the output of:

```
SELECT Genre, AVG(Price)
FROM Book
GROUP BY Genre
ORDER BY Genre;
```

1. Adventure: Holes 12.00, so the average is 12.00.
2. Classic: Emma 6.00 and Rebecca 7.00, average $(6.00 + 7.00) / 2 = 6.50$.
3. SciFi: Dune 8.50 and Neuromancer 10.50, average $19.00 / 2 = 9.50$.
4. Sorted alphabetically by Genre.

Answer: Adventure 12.00; Classic 6.50; SciFi 9.50

Filter, group, sort.

- ✓ **Order:** SELECT, FROM, (INNER JOIN ... ON), WHERE, GROUP BY, ORDER BY.
- ✓ **Aggregates:** SUM, COUNT, AVG; every other SELECTed field must be in GROUP BY.
- ✓ **Joins:** ON foreign key = primary key; use Table.Field names.
- ✓ **Maintenance:** INSERT INTO ... VALUES; UPDATE ... SET ... WHERE; DELETE FROM ... WHERE.

Watch out!

Every field in the SELECT list that is not inside SUM, COUNT or AVG must appear in the GROUP BY. SELECT Title, COUNT(BookID) ... GROUP BY Genre is wrong.

An INNER JOIN only outputs records that have a match. A book that has never been lent does not appear in a join of BOOK and LOAN at all.

UPDATE or DELETE without a WHERE clause changes or deletes every record in the table. Always check the condition, and use the primary key when only one record should change.