

Databases

AS

Read online at pastopic.com/cambridge/computer-science-9618/notes/databases

Last checked 30 September 2026

© 2026 Pastopic. Free for personal study. Please share the link, not the file.

What you need to know

By the end of this topic you should be able to:

1. **Explain the limitations of storing and retrieving data with a file-based approach.**
2. **Describe the features of a relational database that overcome those limitations.**
3. **Understand and use the terms of the relational model:** entity, table, record, field, tuple, attribute, primary key, candidate key, secondary key, foreign key, relationship (one-to-one, one-to-many, many-to-many), referential integrity and indexing.
4. **Use an entity-relationship (E-R) diagram** to record a database design.
5. **Explain the normalisation process** through First, Second and Third Normal Form (1NF, 2NF, 3NF).
6. **Explain why a set of tables is, or is not, in 3NF.**
7. **Produce a normalised design** from a description, a set of data or a set of tables.
8. **Describe what a Database Management System (DBMS) provides** to solve the problems of file-based storage: data management (including the data dictionary), data modelling, the logical schema, data integrity and data security (backups and access rights for individuals and groups).
9. **Explain how the DBMS's software tools are used:** the developer interface and the query processor.
10. **Know that the DBMS creates and changes the database structure with its Data Definition Language (DDL)**, and runs queries and maintains the data with its Data Manipulation Language (DML), and that **SQL** is the industry standard for both.
11. **Read and write SQL DDL:** `CREATE DATABASE`, `CREATE TABLE` with the data types `CHARACTER`, `VARCHAR (n)`, `BOOLEAN`, `INTEGER`, `REAL`, `DATE` and `TIME`, `ALTER TABLE`, `PRIMARY KEY (field)` and `FOREIGN KEY (field) REFERENCES Table (Field)`.
12. **Read and write SQL DML on at most two tables:** `SELECT ... FROM`, `WHERE`, `ORDER BY`, `GROUP BY`, `INNER JOIN`, `SUM`, `COUNT`, `AVG`, and `INSERT INTO`, `DELETE FROM` and `UPDATE`.

Understand it

Why not just use files?

In a **file-based approach**, each program or department keeps its own data file. A sports centre might have one file of members for the bookings desk and another for the accounts office. This causes problems:

- **Data redundancy:** the same data (a member's name and address) is stored in several files, wasting space.
- **Data inconsistency:** when a member moves house, one file gets updated and the other doesn't, so the two copies disagree.
- **Program-data dependence:** each program is written around the exact layout of its file. Add a field to the file and every program that reads it has to be rewritten.
- **Hard to query across files:** answering "which members owe money and have a booking this week?" means writing a new program to match up two files.
- **Hard to share safely:** there is no record locking, so if two users open the same file at once, one user's changes can overwrite the other's.

(Examiners warn against writing "data dependency" when you mean **program-data dependency**.)

What a relational database does instead

A **relational database** stores data in **tables** that are **linked** to each other. Each fact is stored **once**, in one table, and other tables refer to it by its key. So:

- redundancy is reduced, because each item of data is stored only once;
- consistency (integrity) improves, because a change is made in one place and every linked record sees it, and **referential integrity** can be enforced;
- the DBMS sits between programs and the data, giving **program-data independence**: the structure can change without rewriting programs;
- **complex queries** across tables are easy with SQL;
- different users can be given different **views** of the data, so each sees only the part they need;
- many users can work at once, with **record locking** stopping two of them changing the same record at the same time.

The words you must use

A running example: a bike hire shop, with a database called BIKEHIRE.

```
CUSTOMER(CustomerID, FirstName, LastName, Phone)
BIKE(BikeID, Model, FrameSize, DailyRate)
HIRE(HireID, CustomerID, BikeID, StartDate, Days, Returned)
```

On the exam paper the primary key is **underlined**. Here the first field of each table is the primary key; in HIRE, CustomerID and BikeID are foreign keys.

Term	Meaning	In BIKEHIRE
Entity	A real-world thing (object, person, event) that data is stored about; each entity becomes a table	a customer, a bike, a hire
Table	The data about one entity type, in rows and columns	CUSTOMER
Record / tuple	One row of a table: all the data about one instance of the entity	the row for customer C101
Field / attribute	One column of a table: one item of data about the entity	Phone
Primary key	The attribute (or smallest set of attributes) whose value is unique for every record, chosen to identify each record	BikeID in BIKE
Candidate key	Any attribute, or smallest set of attributes, that could be the primary key because it is unique for every record	a bike's frame number, if stored
Secondary key	A candidate key that was not chosen as the primary key, used as an alternative way to search for records	Phone, if every customer's is different
Composite key	A primary key made of two or more attributes	(StudentID, CourseCode) below
Foreign key	An attribute in one table that links to the primary key of another table	CustomerID in HIRE
Relationship	A link between two tables, made by a foreign key; one-to-one, one-to-many or many-to-many	CUSTOMER to HIRE is one-to-many
Referential integrity	Every foreign key value must match an existing primary key value, so linked data stays consistent	HIRE can't hold a CustomerID that isn't in CUSTOMER
Index	A separate sorted list of a field's values with pointers to their records, so the DBMS finds records quickly without reading the whole table	an index on LastName

An entity is **not** the table itself and **not** a field: it is the thing in the real world the table is about. A tuple is a **row**, never a column; examiners see the two mixed up every year.

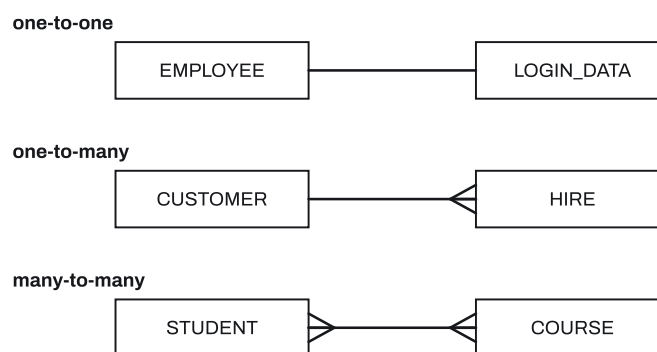
Relationships and E-R diagrams

A relationship is **one-to-many (1:M)** when one record in the first table can link to many in the second: one customer makes many hires, but each hire is by one customer. The link is made by putting the **primary key of the "one" side as a foreign key in the "many" side** (CustomerID goes into HIRE).

One-to-one (1:1): each record links to at most one in the other table, such as an employee and their login details. Put either table's primary key into the other as a foreign key.

Many-to-many (M:M): a student takes many courses and a course has many students. This **cannot be built directly** from two tables, because a single foreign key field can hold only one value. Fix it with a **linking table** whose primary key is made of the two tables' primary keys: STUDENT_COURSE(StudentID, CourseCode). The M:M becomes **two one-to-many relationships**, each pointing at the linking table.

An **entity-relationship (E-R) diagram** documents the design: one box per entity, one line per relationship, and a "crow's foot" at the **many** end.



To complete an E-R diagram from a list of tables, follow every foreign key: draw a line from the table where it is the **primary key** (the "one" end, plain) to the table where it is a **foreign key** (the "many" end, crow's foot). Tables with no foreign key between them get no line.

Normalisation

Normalisation splits data into tables so that each fact is stored once and every non-key field depends on the key. It is done in three steps. A school keeps this unnormalised data about students and the courses they take:

StudentID	StudentName	TutorGroup	TutorName	CourseCode	CourseTitle	Grade
S01	Priya Nair	12A	Mr Lee	CS, MA	Computer Science, Mathematics	A, B
S02	Tom Hart	12B	Ms Diaz	MA, PH	Mathematics, Physics	C, A
S03	Lina Sato	12A	Mr Lee	CS	Computer Science	B

First Normal Form (1NF): no repeating groups, every field atomic, a primary key. The course fields hold lists, a repeating group. Give each student-course pair its own row, split StudentName into FirstName and LastName so each field holds one value, and choose a key. No single field is unique now, so the key is the **composite key** (StudentID, CourseCode). The table has 5 rows:

STUDENT_COURSE(StudentID, CourseCode, FirstName, LastName, TutorGroup, TutorName, CourseTitle, Grade)

Second Normal Form (2NF): in 1NF, and no partial dependencies. A **partial dependency** is a non-key field that depends on only **part** of a composite key. FirstName, LastName, TutorGroup and TutorName depend on StudentID alone; CourseTitle depends on CourseCode alone. Only Grade needs both. Move each group into its own table:

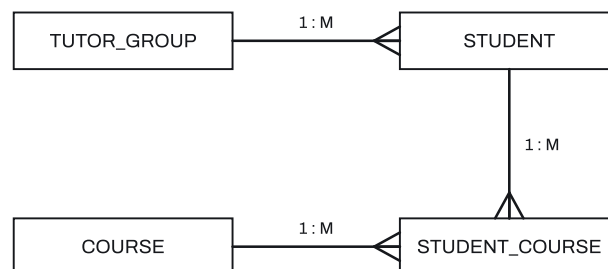
```
STUDENT(StudentID, FirstName, LastName, TutorGroup, TutorName)
COURSE(CourseCode, CourseTitle)
STUDENT_COURSE(StudentID, CourseCode, Grade)
```

A table whose primary key is a single field is automatically free of partial dependencies.

Third Normal Form (3NF): in 2NF, and no non-key (transitive) dependencies. A **non-key dependency** is a field that depends on another non-key field rather than on the key. TutorName depends on TutorGroup: every student in 12A has Mr Lee. Move it out:

```
TUTOR_GROUP(TutorGroup, TutorName)
STUDENT(StudentID, FirstName, LastName, TutorGroup)
COURSE(CourseCode, CourseTitle)
STUDENT_COURSE(StudentID, CourseCode, Grade)
```

Now every non-key field depends on "the key, the whole key and nothing but the key". The many-to-many between students and courses has been replaced by the linking table STUDENT_COURSE.



The Database Management System (DBMS)

A **DBMS** is the software that creates, manages and controls access to the database. Its features address the file-based problems:

- **Data management:** stores, retrieves and updates the data, and keeps a **data dictionary**. The data dictionary is **metadata**, data about the data, not the data itself: table names, field names, data types, field sizes, validation rules, primary and foreign keys, relationships, views.
- **Data modelling:** building a design of the data, its entities and relationships, for example as an E-R diagram.
- **Logical schema:** the overall design of the database's structure (tables, fields, keys, relationships) **independent of any particular DBMS** or how the data is physically stored. It is used to build the physical database.
- **Data integrity:** keeps the data **consistent** by enforcing referential integrity (with **cascading update and delete**, so a changed or deleted primary key updates or removes its linked records and no "orphan" records are left), validation rules, and record locking. Integrity means consistent and up to date; it does **not** guarantee the data is correct.
- **Data security:** usernames and passwords, **access rights** (privileges) that give individuals or groups read-only, read/write or no access to particular tables, **views** that show each user only part of the database, **encryption**, and regular **backups** stored elsewhere so data can be recovered. Files can also be given access rights, so access rights on their own are not an advantage of a relational database over files.

Two DBMS software tools:

- The **developer interface** lets the designer build the database without writing everything in code: create, modify and delete tables and relationships, and design **forms** for input (with drop-down boxes and buttons), **reports** for output, and **menus**.
- The **query processor** takes a query (for example in SQL, or criteria entered by the user), **searches** the database for the matching data, and **organises and returns** the results. "It processes queries" is too vague to earn a mark.

DDL and DML

The DBMS does all of its work through two languages. The **Data Definition Language (DDL)** creates and changes the **structure** (databases, tables, fields, keys). The **Data Manipulation Language (DML)** queries and maintains the **data** (select, insert, update, delete). **SQL** (Structured Query Language) is the industry standard for both.

Data types you need for `CREATE TABLE`:

Type	Holds	Example use
CHARACTER	a single character	a grade 'A'
VARCHAR(<i>n</i>)	text of up to <i>n</i> characters	names, IDs such as B04, phone numbers
BOOLEAN	TRUE or FALSE	Returned
INTEGER	whole numbers	Days, FrameSize
REAL	numbers with a decimal part	DailyRate
DATE	a date	StartDate
TIME	a time	a start time 14:30

An ID made of digits with a leading zero (0123) or with letters is **text**, not an integer, because storing it as a number would lose the zero. A phone number is text too: you never do arithmetic on it.

Key facts and formulas

This topic has no exam formula list. "Learn it" means you must remember it.

Fact or syntax	Status
CREATE DATABASE Name;	Learn it
CREATE TABLE T(Field TYPE, ..., PRIMARY KEY (Field), FOREIGN KEY (Field) REFERENCES Other(Field));	Learn it
ALTER TABLE T ADD Field TYPE; and ALTER TABLE T ADD FOREIGN KEY (Field) REFERENCES Other(Field);	Learn it
SELECT fields FROM T WHERE condition ORDER BY field [ASC or DESC];	Learn it
Join: FROM A INNER JOIN B ON A.Key = B.Key Or FROM A, B WHERE A.Key = B.Key	Learn it
COUNT(field), SUM(field), AVG(field), named with AS Title; GROUP BY field for "each"	Learn it
INSERT INTO T (fields) VALUES (values); UPDATE T SET F = v, G = w WHERE ...; DELETE FROM T WHERE ...;	Learn it
Text in quotes "C101"; numbers and TRUE/FALSE without quotes; dates as #31/12/2025#	Learn it
1NF: no repeating groups, atomic fields, primary key	Learn it
2NF: 1NF and no partial dependencies	Learn it
3NF: 2NF and no non-key (transitive) dependencies; no many-to-many relationships left	Learn it
1:M is built with the "one" side's primary key as a foreign key in the "many" side; M:M needs a linking table	Learn it

How to do it

In the Paper 1 questions we have tagged for this topic (30 questions, 2021–2025), the types came up this often. All 30 are long questions with several parts, so every one mixes types and the counts add up to more than 30.

Question type	Questions
Write an SQL query	17
Describe DBMS features and tools	13
Write a CREATE TABLE or CREATE DATABASE script	11
Complete an E-R diagram	10
Complete or correct a given SQL script	9
Identify or describe relationships between tables	9
Define database terms	9
Explain normal forms and whether a design is in 3NF	9
Identify keys in given tables	8
Write an ALTER TABLE script	7
Relational database versus file-based approach	7
Produce a normalised design	6
Write INSERT, UPDATE or DELETE	6
Describe DBMS security methods	5

1. Write an SQL query (17 questions)

Usually 3 to 5 marks, one per clause. Build it clause by clause:

1. **SELECT** the fields the question asks for, and nothing else. If it says "the number of", use `COUNT ( field )`; "the total", `SUM ( field )`; "the average", `AVG ( field )`. If it asks for "an appropriate name" or "title", add `AS Name`.
2. **FROM** the table or tables that hold those fields **and** the fields in the conditions.
3. **Join** two tables on the shared key, either way (both earn the mark):
 - `FROM HIRE INNER JOIN CUSTOMER ON HIRE.CustomerID = CUSTOMER.CustomerID`
 - `FROM HIRE, CUSTOMER WHERE HIRE.CustomerID = CUSTOMER.CustomerID`
4. **WHERE** with each condition, joined by `AND` (all must be true) or `OR` (either). Text in quotes, numbers and Booleans without.
5. **GROUP BY** when the question says "each" or "per" (a count or total for each customer).
6. **ORDER BY** field, adding `DESC` for descending (ascending is the default).
7. End with `;`.

Variations you will meet:

- **A year or date range:** `StartDate >= #01/01/2025# AND StartDate <= #31/12/2025#`.
- **"After 9 September"** means `> #09/09/2025#`; "on or after" means `>=`.
- **A field name used in both tables** (the key) must be written with its table, `CUSTOMER.CustomerID`, or the query is ambiguous.
- **Two values of one field:** `WHERE Model = "Trail X" OR Model = "Road Pro"`.
- **ORDER BY a field you don't select** is allowed, as in "in descending order of rating".
- **Wildcards** with `LIKE`: `WHERE BikeID LIKE "B0%"` matches every ID starting B0 (% stands for any characters).

2. Describe DBMS features and tools (13 questions)

1. Name the feature, then say **what it does**, then (if asked) an **example**.
2. Learn one clear sentence for each: data dictionary (metadata: table and field names, data types, validation rules, keys, relationships), logical schema (a DBMS-independent model of the whole structure, such as an E-R diagram), developer interface (create tables, forms, reports, menus), query processor (takes the query, searches, returns organised results), data integrity (referential integrity with cascading update/delete, validation), data security (see type 14).

Variations you will meet:

- A table with the name given and the description missing, or the other way round.
- "Identify three **other** items in a data dictionary", when some are already given: pick ones not listed, such as data types, validation rules, relationships, views.
- A paragraph with gaps and a word bank.
- "Which feature describes the relationship between the data and its structure?": the **logical schema**.

3. Write a CREATE TABLE or CREATE DATABASE script (11 questions)

1. `CREATE TABLE NAME (` with an opening bracket.
2. One line per field: the field name, then a sensible type, then a comma. Look at the sample data: letters or leading zeros mean `VARCHAR(n)`, whole numbers `INTEGER`, decimals `REAL`, yes/no `BOOLEAN`, dates `DATE`, times `TIME`.
3. If the question asks for **constraints**, add `NOT NULL` to the key fields and any field that must be filled in.
4. `PRIMARY KEY (Field)`. For a composite key, `PRIMARY KEY (Field1, Field2)`.
5. If the table has a foreign key and the question expects it, `FOREIGN KEY (Field) REFERENCES Other(Field)`.
6. Close with `);`.

`CREATE DATABASE SHOP;` is a one-mark one-liner: the database, not a table.

4. Complete an E-R diagram (10 questions)

1. For each table, find each foreign key: a field that is the primary key of **another** table.
2. Draw a line between those two tables, with the **crow's foot at the table holding the foreign key** (the many end).
3. A linking table gets **two** lines, each with the crow's foot at the linking table.
4. Draw nothing else: extra relationships lose marks.

For example, in BIKEHIRE, HIRE holds CustomerID and BikeID, so there are two relationships: CUSTOMER to HIRE (1:M) and BIKE to HIRE (1:M).

5. Complete or correct a given SQL script (9 questions)

1. Work out what the finished query must return, then read the script clause by clause as in type 1.
2. For gaps, fill in exactly what goes there: a function name (`COUNT`, `SUM`), a table name, a join condition, `GROUP BY`, a `LIKE` pattern.
3. For "find the errors", check each of these in turn:
 - the right function: `COUNT` counts records, `SUM` adds up a numeric field;
 - both tables listed in `FROM`;
 - key fields written with their table name in the join, `STUDENT.StudentID = LESSON.StudentID`;
 - `AND` where both conditions must hold, not `OR`;
 - text values in quotes.
4. Keep the script's own style: if it joins with `WHERE`, don't rewrite it with `INNER JOIN`. Examiners report students losing time this way.

6. Identify or describe relationships between tables (9 questions)

1. **State the degree:** one-to-many, one-to-one or many-to-many, and which way round ("one customer has many hires").
2. **Say how it is built:** "the primary key CustomerID in CUSTOMER is a foreign key in HIRE".

Variations you will meet:

- **Tables described in words only:** decide each pair's degree from the story, then give the key link for each. A table like `JOB_EMPLOYEE` that "stores the employees working on each job" is a linking table: two 1:M relationships.
- **"Explain the purpose of the linking table":** it removes the many-to-many relationship between the two tables, so each record of one can link to many of the other.

- **Which kind can't be built directly?** Many-to-many.

7. Define database terms (9 questions)

Give one precise sentence per term (use the table in "Understand it") and, if the question gives tables, an example from them. For **referential integrity** (often 3 marks), give several points: every foreign key has a matching primary key; related data stays consistent; changes and deletions cascade to linked records; no orphan records. Then apply it: "CompanyID in PLACEMENT must match a CompanyID in COMPANY; deleting a company deletes or blocks its placements."

8. Explain normal forms and whether a design is in 3NF (9 questions)

1. Learn the three rules exactly: 1NF no repeating groups (atomic fields, a key); 2NF no partial dependencies; 3NF no non-key dependencies.
2. For "explain why this database is in 3NF", apply each rule **to the given tables**: "no field holds a list, so there are no repeating groups; the linking table's only non-key field, Grade, depends on both parts of its key, so there are no partial dependencies; in COURSE, CourseTitle depends only on CourseCode, so there are no non-key dependencies; the many-to-many has been replaced by a linking table."
3. For "put this table into 1NF", remove the repeating group (one row per item), split non-atomic fields (a full name into first and last), and choose a primary key.

Variations you will meet:

- **Match each normal form to its definition:** 1NF no repeating groups, 2NF no partial dependencies, 3NF all fields fully dependent on the key (no non-key dependencies).
- **"Why can't this data go in that table?"** For example, order items can't be stored in the order table: one order has many items, so the order table would need a repeating group and would not be in 1NF.
- Describe each form, not the steps between them, when that is what is asked.

9. Identify keys in given tables (8 questions)

1. **Primary key:** the field that is unique for every record in that table. For a linking table it is usually the **composite** of the two foreign keys.
2. **Foreign key:** a field that is the primary key of **another** table. Give its table if asked.
3. **Candidate key:** any other field that is also unique, such as a card number, a serial number or an email address.

A linking table needs a composite key because neither foreign key is unique on its own (one student has many courses), but each **pair** occurs only once.

10. Write an ALTER TABLE script (7 questions)

- **Add a field:** `ALTER TABLE HIRE ADD CollectTime TIME;`, with a sensible field name and type.
- **Add a foreign key** to a table that already exists: `ALTER TABLE HIRE ADD FOREIGN KEY (BikeID) REFERENCES BIKE(BikeID);`
- Two new fields: two ADD clauses, or two statements.

11. Relational database versus file-based approach (7 questions)

Give each advantage as a **point plus a reason** (mark schemes award pairs):

- reduced **data redundancy**, because linked tables mean each item is stored only once;
- better **data consistency / integrity**, because data is updated in one place and referential integrity can be enforced;
- **program-data independence**, because the DBMS manages the structure, so changing it doesn't mean rewriting programs;
- **complex queries** across several tables are easy with SQL;
- **views** let each user see only the data they need;
- **concurrent access** by many users, with record locking.

When asked for "one limitation of the file-based approach and how a relational database fixes it", name the limitation (for example, redundancy), then give the fix and why it works.

12. Produce a normalised design (6 questions)

1. List the **entities** in the description: each becomes a table.
2. Give each a **primary key** (make up an ID field if nothing is unique).
3. Put every field in the table it describes. Check nothing depends on a non-key field.
4. For every **many-to-many** ("a user plays many quizzes; a quiz is played by many users"), add a **linking table** holding both primary keys as foreign keys, plus any data about the pair (a score, a quantity, a grade).
5. Write each table as `TABLE (PrimaryKey, Field, ...)`, underline the primary key, and say which fields are foreign keys and which tables they link to.

Variations you will meet:

- **Split a table with repeated data:** a BATCH table repeating Type, Flavour and Size for every batch becomes an ICE_CREAM table plus BATCH with a foreign key to it.
- **Describe the extra tables** for a new requirement and how they link: name each table, its primary key, its foreign keys and the table each links to.

13. Write INSERT, UPDATE or DELETE (6 questions)

- `INSERT INTO CUSTOMER VALUES ("C104", "Sam", "Reid", "0274455667");`: the values in the same order as the fields; you may list the fields first, `INSERT INTO CUSTOMER (CustomerID, FirstName, LastName, Phone) VALUES (...);`.
- `UPDATE HIRE SET Returned = TRUE, Days = 4 WHERE HireID = "H0112";`: several changes separated by commas, one SET.
- `DELETE FROM HIRE WHERE Returned = TRUE;`.

Always include the `WHERE`: without it, UPDATE changes and DELETE removes **every** record.

14. Describe DBMS security methods (5 questions)

Give a **method**, then **how it protects** the data in the question:

- **Access rights (privileges):** each user or group gets read-only, read/write or no access to each table, so only those with permission can see or edit, say, customer data.
- **Passwords / authentication** (including biometrics or two-factor): stop unauthorised people accessing the database.
- **Encryption:** data stolen without the key can't be understood.
- **Views:** users see only the fields and tables they need.
- **Backups** scheduled regularly and stored elsewhere: a copy survives loss or damage.
- **Record locking:** stops two users changing the same record at once, so updates aren't lost.

Worked examples

The bold codes show where marks are earned, as in Cambridge mark schemes. The number is how many marks.

- **B1** mark for a correct result on its own, with no method needed.

All the SQL examples use the BIKEHIRE database from "Understand it", with this data:

CustomerID	FirstName	LastName	Phone
C101	Ana	Silva	0211234567

CustomerID	FirstName	LastName	Phone
C102	Ben	Okafor	0227654321
C103	Mei	Tan	0219988776

BikeID	Model	FrameSize	DailyRate
B01	Trail X	52	18.50
B02	City Glide	48	12.00
B03	Trail X	56	18.50
B04	Road Pro	58	25.00
B05	City Glide	54	12.00

HireID	CustomerID	BikeID	StartDate	Days	Returned
H0109	C101	B02	15/11/2024	5	TRUE
H0110	C101	B04	12/03/2025	2	TRUE
H0111	C102	B02	20/03/2025	1	TRUE
H0112	C101	B01	02/04/2025	3	FALSE
H0113	C103	B05	05/04/2025	1	FALSE
H0114	C101	B03	06/04/2025	2	FALSE
H0115	C102	B04	28/12/2024	4	TRUE

Example 1

A dental practice keeps its patient details in one file for appointments and another file for billing. It plans to move to a relational database.

(a) Describe **two** limitations of the file-based approach and how a relational database overcomes each. [4]

(b) The database will be managed by a DBMS that keeps a data dictionary. State what is meant by a data dictionary and give **two** items stored in it. [3]

(c) Explain how access rights can protect the patients' data. [2]

(a) The same patient's name and address are stored in both files: **data redundancy**. **B1** In a relational database the patient's details are stored once, in a PATIENT table, which the appointment and bill tables link to by PatientID. **B1**

When a patient changes address, only one file may be updated, so the files disagree: **data inconsistency**. **B1** With the data stored once, the change is made in one place and every linked record sees it; referential integrity keeps the links consistent. **B1**

(b) Data about the data in the database (metadata), not the patients' data itself. **B1** For example, field names and their data types **B1**, and validation rules. **B1**

(c) Each user or group is given permissions for each table. **B1** For example, receptionists get read-only access to appointments and no access to medical notes, so only authorised staff can see or change them. **B1**

Example 2

Use BIKEHIRE and its data above.

(a) Identify the **two** foreign keys in HIRE and the table each one references. [2]

(b) Describe the relationship between CUSTOMER and HIRE. [2]

(c) Write an SQL script to define the table HIRE. Include constraints where appropriate. [5]

(d) The shop wants to record the time each bike is collected, for example 09:15. Write an SQL script to add a field to HIRE for this. [2]

(a) CustomerID, which references CUSTOMER. **B1** BikeID, which references BIKE. **B1**

(b) One-to-many: one customer can make many hires, but each hire is made by one customer. **B1** The primary key CustomerID in CUSTOMER is a foreign key in HIRE. **B1**

(c)

```
CREATE TABLE HIRE(
    HireID VARCHAR(5) NOT NULL,
    CustomerID VARCHAR(4) NOT NULL,
    BikeID VARCHAR(3) NOT NULL,
    StartDate DATE,
    Days INTEGER,
    Returned BOOLEAN,
    PRIMARY KEY (HireID),
    FOREIGN KEY (CustomerID) REFERENCES CUSTOMER(CustomerID),
    FOREIGN KEY (BikeID) REFERENCES BIKE(BikeID)
);
```

B1 CREATE TABLE HIRE with brackets and commas; **B1** the ID fields as VARCHAR (they contain letters); **B1** NOT NULL constraints; **B1** Days INTEGER, StartDate DATE, Returned BOOLEAN; **B1** the primary key (the foreign keys earn the same mark if the scheme asks for them).

(d)

```
ALTER TABLE HIRE
ADD CollectTime TIME;
```

B1 ALTER TABLE HIRE; **B1** ADD with a sensible name and the type TIME.

Example 3

Use BIKEHIRE and its data above.

(a) Write an SQL script to return the BikeID and Model of every bike with a frame size of 54 or more, in ascending order of daily rate. **[4]**

(b) Write an SQL script to return the customer ID, first name and last name of each customer who has bikes not yet returned, and how many of their hires are not returned, with a suitable title. **[4]**

(c) Write an SQL script to return the total number of days hired by customer C101 in 2025. **[3]**

(d) Hire H0112 has been returned after 4 days. Write an SQL script to record this. **[3]**

(a)

```
SELECT BikeID, Model
FROM BIKE
WHERE FrameSize >= 54
ORDER BY DailyRate;
```

B1 SELECT the two fields; **B1** FROM BIKE; **B1** the condition >= 54; **B1** ORDER BY DailyRate (ascending is the default; ASC is also fine). It returns B05 City Glide, then B03 Trail X, then B04 Road Pro.

(b)

```
SELECT CUSTOMER.CustomerID, FirstName, LastName, COUNT(HireID) AS NotReturned
FROM CUSTOMER INNER JOIN HIRE
ON CUSTOMER.CustomerID = HIRE.CustomerID
WHERE Returned = FALSE
GROUP BY CUSTOMER.CustomerID;
```

B1 SELECT the fields with COUNT (...) AS NotReturned; **B1** the two tables joined on CustomerID; **B1** Returned = FALSE (no quotes: it is Boolean); **B1** GROUP BY the customer. Result:

CustomerID	FirstName	LastName	NotReturned
C101	Ana	Silva	2
C103	Mei	Tan	1

Ben Okafor has returned everything, so he doesn't appear.

(c)

```
SELECT SUM(Days) AS TotalDays
FROM HIRE
WHERE CustomerID = "C101"
AND StartDate >= #01/01/2025# AND StartDate <= #31/12/2025#;
```

B1 SUM(Days); **B1** FROM HIRE WHERE with the customer (in quotes); **B1** both date conditions. The 2024 hire H0109 is left out, so the total is $2 + 3 + 2 = 7$ days.

(d)

```
UPDATE HIRE
SET Returned = TRUE, Days = 4
WHERE HireID = "H0112";
```

B1 UPDATE HIRE; **B1** SET both fields; **B1** WHERE on the primary key.

Example 4

A swimming club stores data about its members and the galas (competitions) they swim in. Each member has a unique membership number, a name, a date of birth and a squad. Each gala has a date and a venue. A member can swim in many galas and each gala has many members swimming; the club records each member's time at each gala.

(a) Produce a design for this database in Third Normal Form (3NF), using three tables. Give each table in the form `TableName(PrimaryKey, Field1, Field2, ...)` and identify the primary and foreign keys. [6]

(b) Explain why your design is in 3NF. [3]

(c) The linking table has been created without its foreign key to GALA. Write an SQL script to add it. [2]

(a)

```
MEMBER(MemberID, Name, DateOfBirth, Squad)
GALA(GalaID, GalaDate, Venue)
MEMBER_GALA(MemberID, GalaID, SwimTime)
```

B1 MEMBER with MemberID as primary key **B1** and the other member fields; **B1** GALA with a primary key **B1** and its date and venue; **B1** a linking table holding MemberID and GalaID as foreign keys, which link to MEMBER and GALA, and SwimTime; **B1** its composite primary key (MemberID, GalaID).

(b) No field holds a list or repeating group, so it is in 1NF. **B1** The only non-key field in MEMBER_GALA, SwimTime, depends on **both** MemberID and GalaID, and the other tables have single-field keys, so there are no partial dependencies. **B1** No non-key field depends on another non-key field (for example, Venue depends only on GalaID), and the many-to-many between members and galas is replaced by the linking table. **B1**

(c)

```
ALTER TABLE MEMBER_GALA
ADD FOREIGN KEY (GalaID) REFERENCES GALA(GalaID);
```

B1 ALTER TABLE MEMBER_GALA; **B1** ADD FOREIGN KEY (GalaID) REFERENCES GALA(GalaID).

Common mistakes

- **"More secure" or "access rights" as a relational advantage.** Files can have access rights too, so neither earns credit when comparing with a file-based approach, and nor does "more efficient". Say **views** (each user sees only part of the data) or **concurrent access with record locking**.
- **"Data dependency" for program-data dependency.** The file-based drawback is **program-data dependence**: programs must be rewritten when the file's structure changes.
- **Missing quotes around text, or quotes around a Boolean.** `WHERE CustomerID = C101` fails; it must be `"C101"`. But `Returned = FALSE` has no quotes: a Boolean is not text. Examiners report both; missing quotes most often.
- **Mixing up SUM and COUNT.** `COUNT` counts records ("how many hires"); `SUM` adds up a numeric field ("how many days in total").
- **Forgetting the join.** A query on two tables needs both in `FROM` **and** the key condition (`ON` or `WHERE A.Key = B.Key`), with the table name in front of the shared field.
- **Rewriting a given script.** When asked to complete or correct a script that joins with `WHERE`, keep that style; don't switch to `INNER JOIN`.
- **Programming data types in SQL.** Use the syllabus types (`CHARACTER`, `VARCHAR`, `INTEGER`, `REAL`, `BOOLEAN`, `DATE`, `TIME`), not `STRING` from a programming language. An ID like 0123 stored as an integer loses its leading zero and the mark: it must be `VARCHAR(n)`.
- **CREATE TABLE mixed up with CREATE DATABASE.** Examiners report candidates writing one when the question asks for the other. `CREATE DATABASE` makes the empty database; `CREATE TABLE` defines one table's fields and keys.
- **Rows and columns swapped.** A tuple is a row (a record), an attribute is a column (a field). Some answers say "row" and then give a column as the example.
- **An entity is not a table and not an attribute.** It is the real-world thing the data is about.
- **Generic answers.** "Explain why these tables are in 3NF" or "how access rights protect this data" need the given tables and fields named. Reciting the rules alone earns little.
- **Vague DBMS answers.** "The query processor processes queries" earns nothing; say it takes the query, searches the data and returns the results. A data dictionary is data **about** the data, not the data, and not a description of the DBMS.
- **Data integrity is not accuracy, and not security.** Integrity keeps data consistent and up to date; it can't make wrong data right. Security protects data from unauthorised access or loss. "Security keeps the data safe" is too vague for a mark.
- **Extra or back-to-front lines on an E-R diagram.** Only draw relationships made by a foreign key; an extra line caps your mark. Put the crow's foot at the many end (the table holding the foreign key), not the one end.
- **Copying names wrongly.** Use the table and field names exactly as the question spells them, singular or plural.

How to get the marks

- **SQL is marked clause by clause.** Each correct clause earns its mark on its own, so write every clause even if you're unsure of one. Put each clause on its own line so the examiner can see it.
- **Either join style earns the join mark:** `INNER JOIN ... ON` or a `WHERE` condition.
- **"Appropriate name" or "suitable title"** is a mark for `AS: COUNT(OrderID) AS NotCollected`.
- **"Include constraints"** means `NOT NULL` (on keys and required fields) at least once.
- **"Identify"** needs just the name (the key, the table, the relationship). **"Describe"** needs what it is or does. **"Explain"** needs the reason, "because ...". A "describe two methods" question marks in pairs: the method, then how it works.
- **Relationships:** state the degree **and** how it is implemented with the primary and foreign key, naming the fields.
- **Normalised designs:** write every table in `TABLE (Key, ...)` form, underline the primary keys, and say which fields are foreign keys. Use the question's field names and keep every original field.
- **Definitions:** one exact sentence plus, where tables are given, an example from them.
- **Referential integrity** answers score well with three separate points: foreign keys match primary keys, cascading update/delete, no orphan records, then one example from the tables.

Quick check

1. A table STAFF(StaffID, Name, Email, PassportNo) has StaffID as its primary key. Every member of staff has a different email address and a different passport number. Identify the candidate keys and state what the ones not chosen as the primary key are called.
2. This script should return the total number of days hired by customer C102 on hires that have been returned. Find and correct the **three** errors, then give the value it returns with the sample data in the worked examples.

```
SELECT COUNT(Days) AS TotalDays
FROM HIRE
WHERE CustomerID = C102
OR Returned = TRUE;
```

3. Write an SQL script to return the average daily rate of all the bikes in BIKE, with a suitable title, and state the value it returns.
4. The table ORDER_LINE(OrderID, ProductID, ProductName, Quantity) has the composite primary key (OrderID, ProductID). Explain which normal form it breaks and normalise it.
5. Write an SQL script to add a new customer to CUSTOMER: ID C104, Sam Reid, phone number 0274455667.

Answers

1. StaffID, Email and PassportNo are all candidate keys, because each is unique for every record. Email and PassportNo, which were not chosen, are **secondary keys**.

2. COUNT should be SUM (it adds up the days); C102 needs quotes, "C102"; OR should be AND (both conditions must hold). Corrected:

```
SELECT SUM(Days) AS TotalDays
FROM HIRE
WHERE CustomerID = "C102"
AND Returned = TRUE;
```

C102's returned hires are H0111 and H0115, so it returns $1 + 4 = 5$.

3. `SELECT AVG(DailyRate) AS AverageRate FROM BIKE;` It returns $(18.50 + 12.00 + 18.50 + 25.00 + 12.00) \div 5 = 86.00 \div 5 = 17.20$.

4. ProductName depends only on ProductID, which is **part** of the key: a partial dependency, so the table is not in 2NF. Move it into its own table: `PRODUCT(ProductID, ProductName)` and `ORDER_LINE(OrderID, ProductID, Quantity)`, where ProductID in `ORDER_LINE` is a foreign key to `PRODUCT`. Quantity depends on both parts of the key, so it stays.

5. `INSERT INTO CUSTOMER VALUES ("C104", "Sam", "Reid", "0274455667");` The phone number is text, so it goes in quotes and keeps its leading zero.

Past questions to try

- [9618/11/O/N/24 Q2](#): file-based limitations, an E-R diagram, CREATE TABLE with a composite key, and a SUM query.
- [9618/11/M/J/24 Q6](#): a 3NF design from a description, data dictionary and logical schema, ALTER TABLE and GROUP BY.
- [9618/12/M/J/23 Q2](#): DBMS tools, referential integrity, and finding four errors in a script.
- [9618/12/O/N/22 Q5](#): removing a many-to-many relationship, INSERT INTO and candidate keys.