Databases

International Baccalaureate · IB Diploma · Computer Science · SL

Scope and prerequisites

This handout teaches Theme A3, Databases, for the course first assessed in 2027: relational databases, design and normalisation, and SQL.

Prerequisites: data types from B2 (integer, string, Boolean, real).

Why use a database?

■ One spreadsheet, many problems

A school library keeps loans in one spreadsheet: each row has the student's name, form, the book's title and author, and the date. When a student changes form, dozens of rows must be edited, and some are missed. Two rows now disagree about the same student. A database is designed to stop this.

■ Database fundamentals

A **database** is an organised collection of data. A **database management system** (DBMS) is the software that stores it, runs queries, controls access and makes backups.

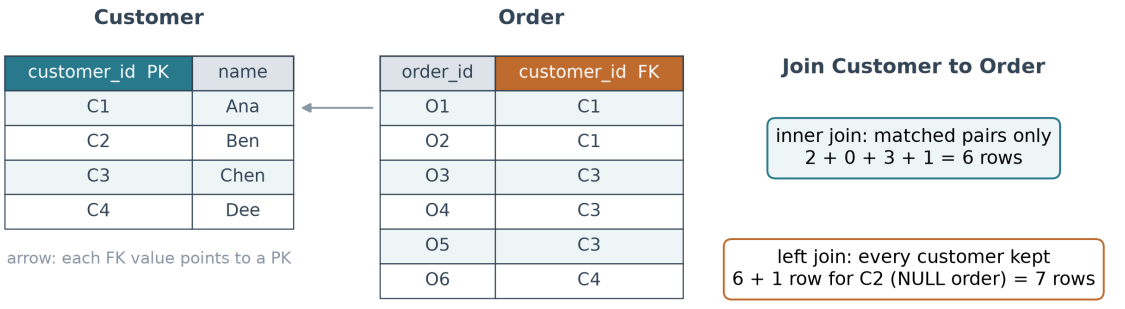
Problem with flat files	How a relational database helps
data redundancy: the same data stored many times	each fact stored once
inconsistency: copies disagree	one copy, so nothing to disagree
hard to search and combine	SQL queries join tables
no control over who sees what	user accounts and permissions

■ Tables, records and keys

A **relational database** stores data in **tables** (relations). Each **record** (row) is one item; each **field** (column, attribute) is one property.

Key	Meaning	Example
primary key	a field whose value is unique for every record	`student_id`
foreign key	a field that holds the primary key of another table, linking them	`student_id` in Loan
composite key	a primary key made of two or more fields	(`order_id`, `item_id`)

Referential integrity requires a non-null foreign key to match a referenced key (a primary or other suitable unique key); nullable foreign keys may represent no specified reference: a loan cannot refer to a student who does not exist.



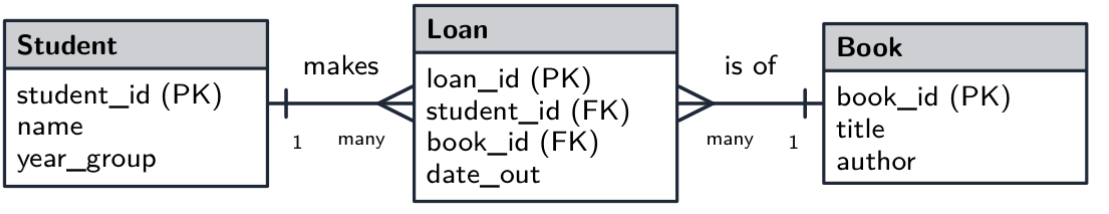
A foreign key value must match an existing primary key; the join type decides which unmatched rows survive.

A foreign key links each order to its customer

Database design

■ Entities and relationships

An **entity** is a thing we store data about: a student, a book. An **entity-relationship diagram** (ERD) shows the entities and how they are related.



One student makes many loans; each loan is of one book

Relationship	Meaning	Example
one-to-one	each A has one B	a student and their locker
one-to-many	one A has many B	one student, many loans
many-to-many	many A, many B	students and books

A many-to-many relationship cannot be stored directly: it is split into two one-to-many relationships through a **link table**, here `Loan`.

Normalisation

Normalisation organises tables to remove redundancy. Start from one unnormalised table:

order_id	customer	phone	items
501	Li Na	1381	pen, ruler
502	Li Na	1381	pen

Form	Rule	Result
first normal form (1NF)	every field holds one atomic value; no repeating groups	one row per item: key (`order_id`, `item`)
second normal form (2NF)	1NF, and no non-key field depends on only part of a candidate key	`customer` depends only on `order_id`: move it to an Order table; `price` depends only on `item`: move it to an Item table
third normal form (3NF)	2NF, with no transitive dependency of a non-key field on a candidate key	`phone` depends on the customer, not the order: move it to a Customer table

Result in 3NF:

- Customer(**customer_id**, name, phone)
- Order(**order_id**, customer_id)
- OrderLine(**order_id**, **item_id**, quantity)
- Item(**item_id**, description, price)

Now Li Na's phone number is stored once. A short test for 3NF: every non-key field depends on "the key, the whole key, and nothing but the key".

Choosing data types

Each field needs a type, such as `INTEGER`, `VARCHAR(50)` (text up to 50 characters), `DATE`, `BOOLEAN` or `DECIMAL(6,2)`. Phone numbers are stored as text, not integers: they can start with 0 and are never added up.

SQL

Querying data

SQL (Structured Query Language) defines and queries relational databases. The examples use these tables:

Student

student_id	name	year_group
1	Ana	11
2	Ben	12
3	Chen	11

Book

book_id	title	author
10	Dune	Herbert
11	Emma	Austen
12	Holes	Sachar

Loan

loan_id	student_id	book_id	date_out
100	1	10	2026-09-01
101	1	12	2026-09-05
102	3	11	2026-09-06

```
SELECT name
FROM Student
WHERE year_group = 11
ORDER BY name DESC;
```

Result: Chen, Ana. `WHERE` filters rows; `ORDER BY … DESC` sorts Z to A.

```
SELECT Student.name, Book.title
FROM Loan
JOIN Student ON Loan.student_id = Student.student_id
JOIN Book ON Loan.book_id = Book.book_id;
```

Result: (Ana, Dune), (Ana, Holes), (Chen, Emma). A **join** combines tables by matching the foreign key with the primary key. Ben has no loans, so he does not appear.

```
SELECT student_id, COUNT(*) AS loans
FROM Loan
GROUP BY student_id;
```

Result: (1, 2), (3, 1). **Aggregate functions** such as `COUNT`, `SUM`, `AVG`, `MAX` and `MIN` work on groups of rows.

Clause	Use
`WHERE`	filter rows: `=`, `<>`, `>`, `LIKE 'D%'`, `AND`, `OR`
`ORDER BY`	sort, `ASC` or `DESC`
`GROUP BY`	make groups for aggregate functions
`HAVING`	filter groups: `HAVING COUNT(*) > 1`

Changing data

```
INSERT INTO Book (book_id, title, author) VALUES (13, 'Wonder',
↪  'Palacio');
UPDATE Student SET year_group = 12 WHERE student_id = 3;
DELETE FROM Loan WHERE loan_id = 100;
```

An `UPDATE` or `DELETE` without `WHERE` changes **every** row.

Transactions

A **transaction** is a group of operations that must all succeed or all fail, such as moving money: take it from one account and add it to another. Transactions follow **ACID**:

Property	Meaning
atomicity	all of it happens, or none of it
consistency	the database moves from one valid state to another
isolation	transactions running at once do not interfere
durability	once committed, a change survives a crash

Practice

1. State **two** problems of storing a school's data in one flat file.
2. Define primary key and foreign key, with an example from the library tables.
3. Explain why the many-to-many relationship between students and books needs the Loan table.

4. A table has fields (`booking_id`, `room`, `room_capacity`, `date`). Explain why it is not in 3NF and normalise it.
5. Write SQL to list the titles of all books by Austen.
6. Write SQL to list the names of students in year 11 with the number of loans each has made.
7. State the result of `SELECT COUNT(*) FROM Loan WHERE book_id > 10;` for the tables above.
8. Write SQL to change the author of book 12 to 'L. Sachar'.
9. Explain atomicity using a bank transfer.

Answers.

1. E.g. data redundancy, the same student stored in many rows; inconsistency when one copy is updated and others are not; hard to search or control access.
2. A primary key uniquely identifies each record, such as `book_id` in Book. A foreign key is a field holding another table's primary key, such as `book_id` in Loan.
3. One student borrows many books and one book is borrowed by many students. A relational table cannot store this directly; the link table turns it into two one-to-many relationships, each loan linking one student and one book.
4. `room_capacity` depends on `room`, a non-key field: a transitive dependency. Split into Booking(**booking_id**, room, date) and Room(**room**, room_capacity).
5. `SELECT title FROM Book WHERE author = 'Austen';`
6. `SELECT Student.name, COUNT(*) FROM Student JOIN Loan ON Student.student_id = Loan.student_id WHERE Student.year_group = 11 GROUP BY Student.name;` giving (Ana, 2), (Chen, 1).
7. Loans 101 (book 12) and 102 (book 11): **2**.
8. `UPDATE Book SET author = 'L. Sachar' WHERE book_id = 12;`
9. The money must both leave one account and arrive in the other; if the system fails part-way, the whole transaction is undone, so money is never lost or created.
10. A graph database: people are nodes and friendships are links, so queries such as "friends of friends" follow links directly.
11. E.g. a warehouse holds historical data from many sources, an operational database current data; a warehouse is mostly read for large analysis queries, an operational database handles many small updates.

Exam tips

- Keys: say the primary key is **unique** for each record; a foreign key **links** to a primary key in another table.
- In normalisation answers, name the dependency you are removing, then show the new tables with their keys.

- Write SQL keywords in capitals, put text in single quotes, and end each statement with a semicolon.
- In a join, always give the `ON` condition matching the foreign key to the primary key.
- Check `UPDATE` and `DELETE` statements for a `WHERE` clause.
- Learn ACID as four words with one-line meanings, and illustrate with a transfer.