SQL Handout

English edition

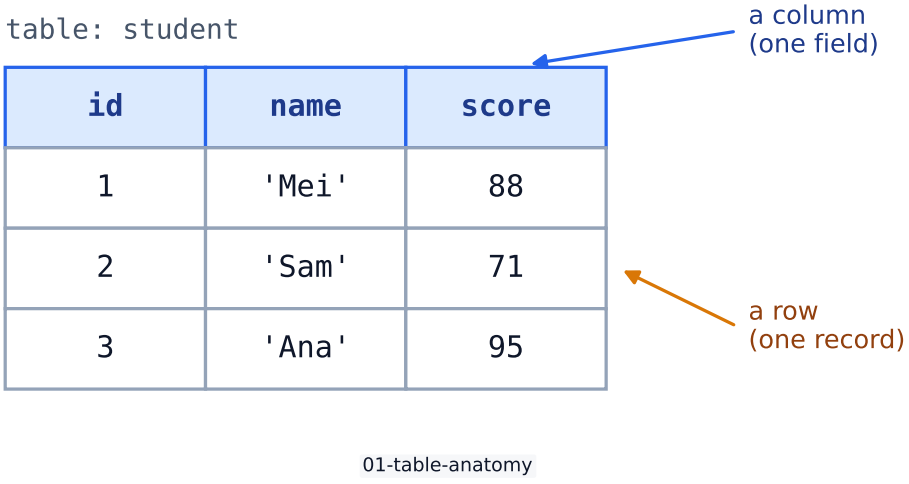
Contents

1 Querying basics

1.1 SELECT and FROM

A database 数据库 keeps data in tables 表. A query 查询 reads data with `SELECT`. `SELECT *` returns every column; `FROM` names the table.

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1, 'Mei', 88), (2, 'Sam', 71);
SELECT * FROM student;
```



A table stores records: each row is one record, each column is one field

- Each row 行 is one record 记录. SQL keywords are written in UPPERCASE by habit.
- A statement ends with a semicolon `;`.

1.2 Choosing columns

List the columns 列 you want, separated by commas. Use `AS` to rename a column in the result (an alias 别名).

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1, 'Mei', 88), (2, 'Sam', 71);
SELECT name, score AS mark FROM student;
```

A selected column can also be a calculation — pair it with `AS` to name the new column:

```sql
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1, 'Mei', 88), (2, 'Sam', 71);
SELECT name, score + 5 AS bonus FROM student;
```

- `SELECT DISTINCT col` removes duplicate values from the result.

1.3 Filtering with WHERE

`WHERE` keeps only the rows that match a condition 条件. Compare with `=`, `<>` (not equal), `<`, `>`, `<=`, `>=`. Put text in single quotes.

```sql
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1, 'Mei', 88), (2, 'Sam', 71), (3, 'Ana',
↪  95);
SELECT name, score FROM student WHERE score >= 80;
```

- SQL runs the parts in this order: `FROM` → `WHERE` → `SELECT`.

Common mistakes

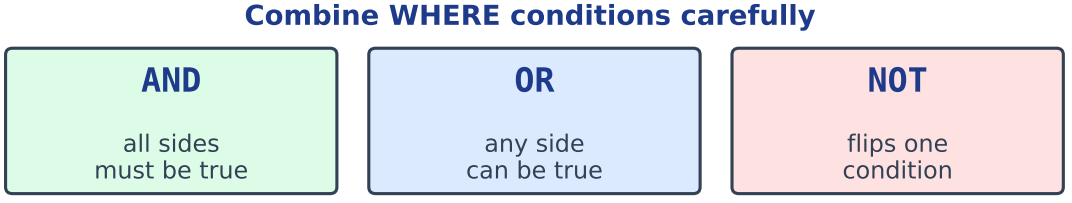
- Put text values in single quotes: `WHERE name = 'Ann'`; column names have no quotes.
- `SELECT *` returns every column — name only the columns you need.
- `WHERE` filters rows; it comes after `FROM`.

2 Filtering & logic

2.1 AND, OR, NOT

Combine conditions with `AND`, `OR`, `NOT`. `AND` needs all sides true; `OR` needs any side true; `NOT` reverses one. Use brackets to set the precedence 优先级.

```
CREATE TABLE student (id INTEGER, name TEXT, form TEXT, score INTEGER);
INSERT INTO student VALUES
↪  (1,'Mei','11A',88),(2,'Sam','11B',71),(3,'Ana','11A',95);
SELECT name FROM student WHERE form = '11A' AND score >= 90;
```



Exam tip: use brackets for precedence; AND is stricter than OR

AND needs all true; OR needs any; NOT flips one

2.2 LIKE, IN and BETWEEN

`LIKE` matches a text pattern 模式: `%` stands for any text and `_` for one character — these are wildcards 通配符.

```
CREATE TABLE student (id INTEGER, name TEXT, form TEXT, score INTEGER);
INSERT INTO student VALUES
↪  (1,'Mei','11A',88),(2,'Sam','11B',71),(3,'Ana','11A',95);
SELECT name FROM student WHERE name LIKE 'M%';        -- starts with M
```

Pattern	Matches
`'M%'`	starts with M
`'%a'`	ends with a
`'%an%'`	contains ”an”
`'M_i'`	M, then exactly one character, then i

`IN` matches a set 集合 of values; `BETWEEN` matches a range 范围 (both ends included).

```
CREATE TABLE student (id INTEGER, name TEXT, form TEXT, score INTEGER);
INSERT INTO student VALUES
↪  (1,'Mei','11A',88),(2,'Sam','11B',71),(3,'Ana','11A',95);
SELECT name, score FROM student
WHERE form IN ('11A', '11C') AND score BETWEEN 80 AND 100;
```

2.3 NULL

`NULL` marks a missing value 缺失值 — nothing was stored in that cell. `NULL` is not `0` and not an empty string. A comparison with `=` or `<>` never matches it; test with `IS NULL` or `IS NOT NULL`.

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1, 'Mei', 88), (2, 'Sam', NULL), (3, 'Ana',
↪  95);
SELECT name FROM student WHERE score IS NULL;
```

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1, 'Mei', 88), (2, 'Sam', NULL), (3, 'Ana',
↪  95);
SELECT name, score FROM student WHERE score IS NOT NULL;
```

- `WHERE score = NULL` returns no rows at all — not even Sam's row.
- A NULL in arithmetic gives NULL: `score + 5` stays NULL for Sam.

Common mistakes

- Test for an empty value with `IS NULL`, never `= NULL`.
- In `LIKE`, `%` matches any text and `_` matches one character: `'A%'` means "starts with A".
- `IN (1, 2, 3)` is shorter than many `OR`s; `BETWEEN a AND b` includes both ends.

3 Sorting & limiting

3.1 ORDER BY and LIMIT

`ORDER BY` sorts 排序 the result rows. Add `DESC` for descending 降序 (high to low); the default is ascending 升序 (low to high).

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1,'Mei',88),(2,'Sam',71),(3,'Ana',95);
SELECT name, score FROM student ORDER BY score DESC;
```

3.1.1 Sort by more than one column

List several columns. A tie 平局 in the first column is broken by the next.

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1,'Mei',88),(2,'Sam',88),(3,'Ana',95);
SELECT name, score FROM student ORDER BY score DESC, name ASC;
```

3.1.2 LIMIT (top-N)

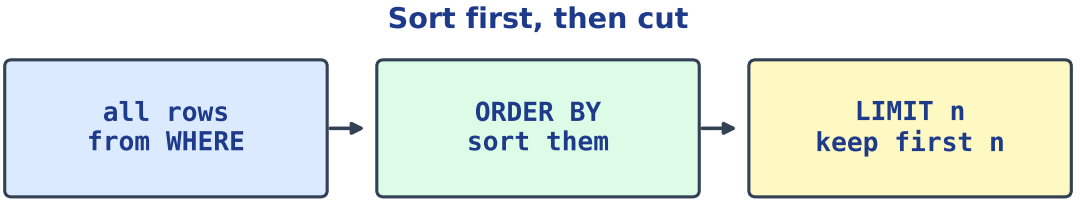
`LIMIT n` keeps only the first `n` rows — pair it with `ORDER BY` for a top-N 前 N 名 list.

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1,'Mei',88),(2,'Sam',71),(3,'Ana',95);
SELECT name, score FROM student ORDER BY score DESC LIMIT 2;
```

- `ORDER BY` can also sort by an alias or a calculation: `ORDER BY avg_score DESC`.
- Text sorts alphabetically: `ORDER BY name` runs A → Z.

Common mistakes

- `ORDER BY` sorts ascending by default; add `DESC` for descending.
- `ORDER BY` comes near the end, after `WHERE` and `GROUP BY`.
- `LIMIT` caps how many rows come back, but only after sorting.



Exam tip: ORDER BY before LIMIT; ASC is default; DESC reverses the order

ORDER BY sorts; LIMIT keeps the first n after sorting

4 Aggregates & grouping

4.1 Aggregate functions

An aggregate function 聚合函数 turns many rows into one summary 汇总 value. Wrap an average in `ROUND(x, 2)` to tidy it.

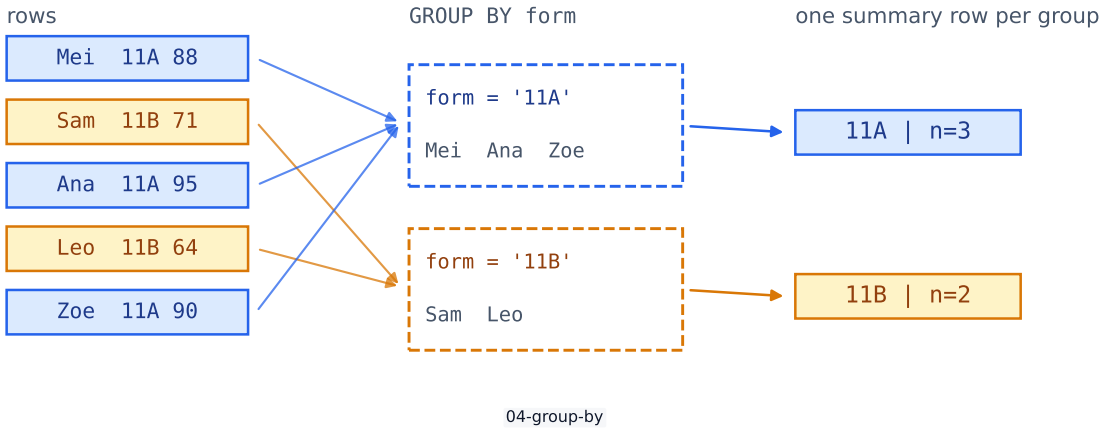
Function	Gives
`COUNT(*)`	how many rows
`SUM(col)`	the total
`AVG(col)`	the mean average
`MIN(col)` / `MAX(col)`	the smallest / largest value

```
CREATE TABLE student (id INTEGER, name TEXT, form TEXT, score INTEGER);
INSERT INTO student VALUES
↪  (1,'Mei','11A',88),(2,'Sam','11B',71),(3,'Ana','11A',95);
SELECT COUNT(*), ROUND(AVG(score), 2), MAX(score) FROM student;
```

4.2 GROUP BY and HAVING

`GROUP BY` makes one summary row per group 组. `HAVING` filters those groups — it is like `WHERE`, but it runs after grouping.

```
CREATE TABLE student (id INTEGER, name TEXT, form TEXT, score INTEGER);
INSERT INTO student VALUES
↪  (1,'Mei','11A',88),(2,'Sam','11B',71),(3,'Ana','11A',95);
SELECT form, COUNT(*) AS n, ROUND(AVG(score), 2) AS avg_score
FROM student
GROUP BY form
HAVING COUNT(*) > 1;
```



GROUP BY collects rows into one bucket per value; each bucket becomes one summary row

Whatever order you write them in, SQL always runs the clauses of a query in the same fixed order:

Step	Clause
1	`FROM` (and any `JOIN`)
2	`WHERE` — filter rows
3	`GROUP BY` — form groups
4	`HAVING` — filter groups
5	`SELECT` — compute the output columns
6	`ORDER BY`, then `LIMIT`

Common mistakes

- You cannot select a plain column beside an aggregate unless it is in `GROUP BY`.
- Filter rows with `WHERE` (before grouping) and filter groups with `HAVING` (after).

- `COUNT(*)` counts rows; `COUNT(col)` skips NULLs in that column.

5 Joins

5.1 Keys & relationships

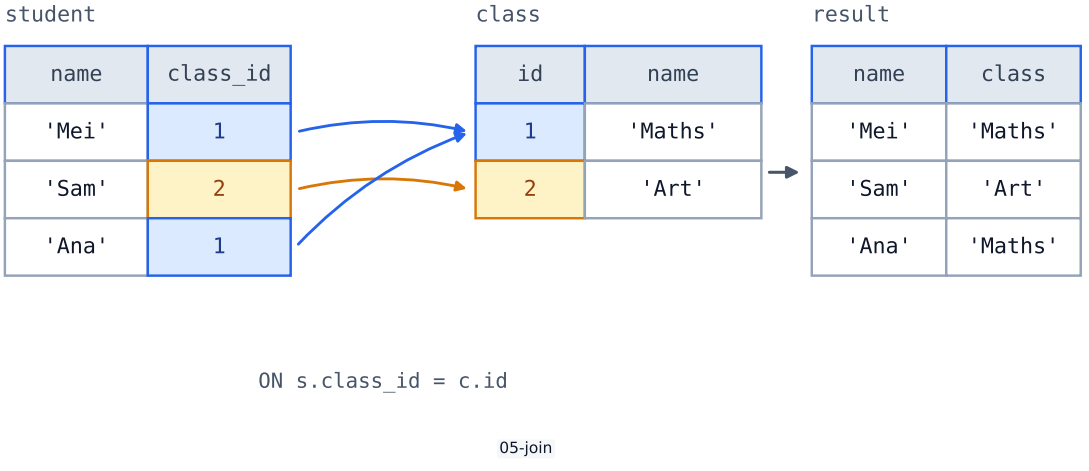
A primary key 主键 uniquely names each row in a table. A foreign key 外键 in one table points to the primary key of another — that builds a relationship 关系 between them.

```
CREATE TABLE class (id INTEGER, name TEXT);
CREATE TABLE student (id INTEGER, name TEXT, class_id INTEGER);
INSERT INTO class VALUES (1, 'Maths'), (2, 'Art');
INSERT INTO student VALUES (1, 'Mei', 1), (2, 'Sam', 2);
SELECT * FROM student;
```

5.2 INNER JOIN

A join 连接 combines rows from two tables. `INNER JOIN ... ON ...` keeps rows where the keys match. A short alias 别名 (`s`, `c`) keeps the query readable.

```
CREATE TABLE class (id INTEGER, name TEXT);
CREATE TABLE student (id INTEGER, name TEXT, class_id INTEGER);
INSERT INTO class VALUES (1, 'Maths'), (2, 'Art');
INSERT INTO student VALUES (1, 'Mei', 1), (2, 'Sam', 2);
SELECT s.name, c.name AS class
FROM student s INNER JOIN class c ON s.class_id = c.id;
```



INNER JOIN matches each foreign key to a primary key and combines the matched rows

5.3 Joining with grouping

Join first, then `GROUP BY` to summarise across the joined rows.

```
CREATE TABLE class (id INTEGER, name TEXT);
CREATE TABLE student (id INTEGER, name TEXT, class_id INTEGER);
INSERT INTO class VALUES (1, 'Maths'), (2, 'Art');
INSERT INTO student VALUES (1, 'Mei', 1), (2, 'Sam', 2), (3, 'Ana', 1);
SELECT c.name AS class, COUNT(*) AS n
FROM student s INNER JOIN class c ON s.class_id = c.id
GROUP BY c.name;
```

5.4 LEFT JOIN

An INNER JOIN keeps only matched rows. A `LEFT JOIN` 左连接 keeps **every** row of the left (first) table; where there is no match, the right table's columns come back as `NULL`.

```
CREATE TABLE class (id INTEGER, name TEXT);
CREATE TABLE student (id INTEGER, name TEXT, class_id INTEGER);
INSERT INTO class VALUES (1, 'Maths'), (2, 'Art');
INSERT INTO student VALUES (1, 'Mei', 1), (2, 'Sam', NULL);
SELECT s.name, c.name AS class
FROM student s LEFT JOIN class c ON s.class_id = c.id;
```

- Sam has no class, but the row still appears — with `class` as `NULL`.
- To find only the unmatched rows, add `WHERE c.id IS NULL`.

Common mistakes

- A join with no `ON` condition pairs every row with every row (a cross join).
- Match the foreign key to the primary key: `ON orders.customer_id = customers.id`.
- An INNER JOIN drops rows that have no match on the other side; use a LEFT JOIN to keep them.

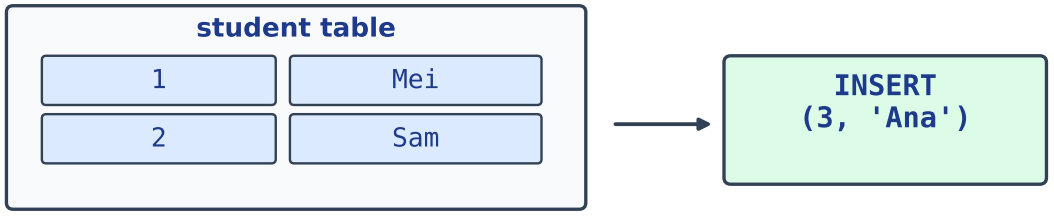
6 Modifying data

6.1 INSERT

`INSERT INTO ... VALUES ...` adds new rows 行. Name the columns, then give the values in the same order. (Each block below ends with a `SELECT` so you can see the result.)

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student (id, name, score) VALUES (1, 'Mei', 88);
INSERT INTO student VALUES (2, 'Sam', 71);
SELECT * FROM student;
```

- `INSERT INTO t VALUES (...), (...);` adds several rows in one statement.



Exam tip: INSERT INTO table VALUES (...); column order must match

INSERT adds a new row to a table

6.2 UPDATE

`UPDATE ... SET ... WHERE ...` changes existing rows. **Always** add `WHERE`, or every row changes.

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1, 'Mei', 88), (2, 'Sam', 71);
UPDATE student SET score = 75 WHERE name = 'Sam';
SELECT * FROM student;
```

6.3 DELETE

`DELETE FROM ... WHERE ...` removes rows. Without `WHERE` it empties the whole table 表.

```
CREATE TABLE student (id INTEGER, name TEXT, score INTEGER);
INSERT INTO student VALUES (1, 'Mei', 88), (2, 'Sam', 71);
DELETE FROM student WHERE score < 80;
SELECT * FROM student;
```

Common mistakes

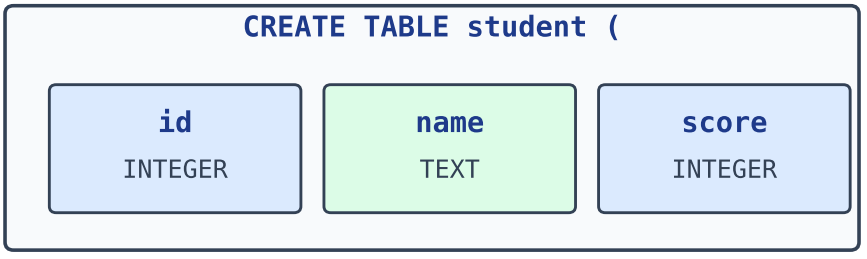
- `UPDATE` and `DELETE` **without** a `WHERE` change every row — always add the `WHERE`.
- In `INSERT`, the values must line up with the column list in order and type.
- Test a risky `DELETE` first as a `SELECT` with the same `WHERE`.

7 Defining tables

7.1 CREATE TABLE & data types

`CREATE TABLE` defines a table's schema 表结构: the column names and their data types 数据类型. The main SQLite types are `INTEGER`, `TEXT`, and `REAL` (a decimal number).

```
CREATE TABLE student (id INTEGER, name TEXT, score REAL);
INSERT INTO student VALUES (1, 'Mei', 88.5);
SELECT * FROM student;
```



Exam tip: name columns and types first; PRIMARY KEY marks the unique id

CREATE TABLE names columns and their types

7.2 Keys & constraints

A constraint 约束 is a rule on a column: `PRIMARY KEY` (a unique id), `NOT NULL` (must have a value), `UNIQUE`, and `DEFAULT` (a fallback value).

```
CREATE TABLE student (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    score INTEGER DEFAULT 0
);
INSERT INTO student (id, name) VALUES (1, 'Mei');
SELECT * FROM student;
```

Declare a foreign key with `REFERENCES` — it records that the column points at another table's primary key:

```sql
CREATE TABLE class (id INTEGER PRIMARY KEY, name TEXT);
CREATE TABLE student (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    class_id INTEGER REFERENCES class(id)
);
INSERT INTO class VALUES (1, 'Maths');
INSERT INTO student VALUES (1, 'Mei', 1);
SELECT s.name, c.name AS class
FROM student s INNER JOIN class c ON s.class_id = c.id;
```

- The long form is `FOREIGN KEY (class_id) REFERENCES class(id)` on its own line.

7.3 ALTER TABLE

`ALTER TABLE ... ADD COLUMN ...` changes the schema of a table that already exists. A `DEFAULT` fills the new column in the old rows.

```sql
CREATE TABLE student (id INTEGER, name TEXT);
INSERT INTO student VALUES (1, 'Mei');
ALTER TABLE student ADD COLUMN score INTEGER DEFAULT 0;
SELECT * FROM student;
```

- `ALTER TABLE student RENAME TO pupil;` renames the whole table.
- `DROP TABLE student;` deletes the table completely — structure **and** data.

Common mistakes

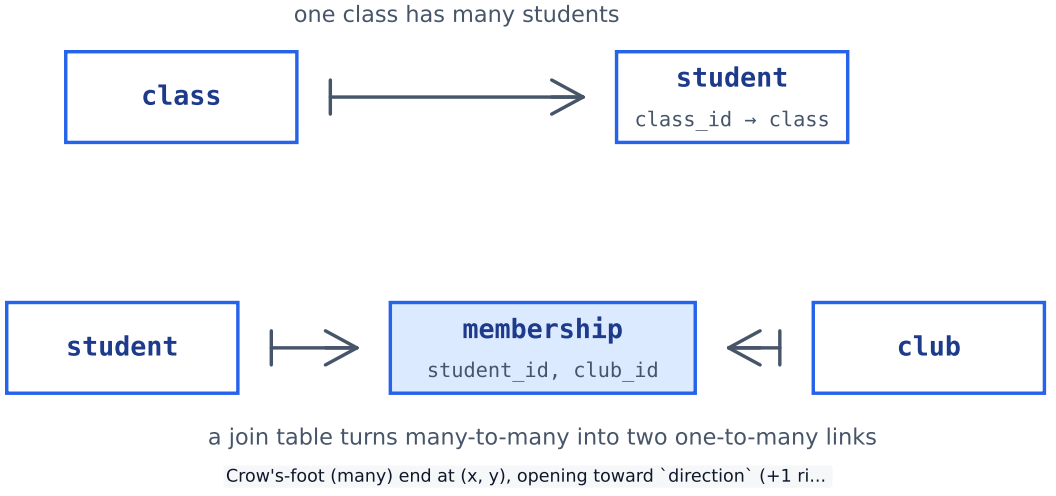
- Every column needs a data type (e.g. `INTEGER`, `TEXT`).
- A PRIMARY KEY must be unique and cannot be NULL.
- A foreign key value must exist in the table it points to.

8 Database design

8.1 Relationships & ER diagrams

Tables connect through relationships 关系. One-to-many 一对多 is the most common: one class has many students. Many-to-many 多对多 needs a join table 连接表 in the middle. An entity-relationship diagram 实体关系图 (ER diagram) draws each entity 实体 as a box and each relationship as a line.

Relationship	Example
one-to-one	a person and their passport
one-to-many	a class and its students
many-to-many	students and clubs



The "many" side carries the foreign key; a many-to-many link needs a join table

The join table holds one row per link. Here Mei is in two clubs, and Chess has two members:

```
CREATE TABLE student (id INTEGER PRIMARY KEY, name TEXT);
CREATE TABLE club (id INTEGER PRIMARY KEY, name TEXT);
CREATE TABLE membership (student_id INTEGER, club_id INTEGER);
INSERT INTO student VALUES (1, 'Mei'), (2, 'Sam');
INSERT INTO club VALUES (1, 'Chess'), (2, 'Art');
INSERT INTO membership VALUES (1, 1), (1, 2), (2, 1);
SELECT s.name, c.name AS club
FROM membership m
INNER JOIN student s ON m.student_id = s.id
```

```
INNER JOIN club c ON m.club_id = c.id;
```

8.2 Normalisation

Normalisation 范式化 organises tables to avoid redundancy 冗余 (the same data repeated) and the update mistakes it causes. The first three normal forms 范式:

- **1NF**: every cell holds one atomic 原子 value — no lists inside a cell.
- **2NF**: no column depends on only part of a composite key 复合主键.
- **3NF**: no column depends on another non-key column.

Unnormalised (bad)	Normalised (better)
`student(name, club1, club2)`	`student(name)` + `membership(student, club)`

Common mistakes

- Split repeating groups into their own table (normalisation) instead of many similar columns.
- Each table should describe ONE kind of thing.
- Link tables with a foreign key that points to another table's primary key.