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;
```

- 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;
```

- `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.

```
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`.

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;
```

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
```

`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;
```

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;
```

4 Aggregates & grouping

4.1 Aggregate functions

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

```
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;
```

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;
```

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;
```

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;
```

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;
```

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;
```

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;
```

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.

```
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;
```

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

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)`