A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

8.3.1 Joining two tables, aggregates with GROUP BY, and choosing SQL data types

Name: _____ Class: _____ Date: _____ **Total: 30 marks**

KEY WORDS inner join 内连接 • data type 数据类型 • query 查询 • join 连接 • foreign key 外键

Objective

Build the skills to answer the **SQL** questions that sheet 8.3 does not reach. That sheet separates DDL from DML and writes single-table `SELECT`, `INSERT`, `UPDATE` and `DELETE`; this one covers the parts of the syllabus subset it never uses: the **INNER JOIN** 内连接 that queries two tables, the **aggregate functions** 聚合函数 `SUM`, `COUNT` and `AVG` with **GROUP BY**, choosing an appropriate **data type** 数据类型 for every attribute, and `CREATE DATABASE`.

You must be able to:

- create a table definition with attributes of appropriate data types: `CHARACTER`, `VARCHAR(n)`, `BOOLEAN`, `INTEGER`, `REAL`, `DATE`, `TIME`
- create a database, add a **primary key** 主键 and add a **foreign key** 外键, and change a table definition with `ALTER TABLE`
- write an SQL script to query data held in at most **two** tables, using `INNER JOIN`
- use `SUM`, `COUNT`, `AVG`, `GROUP BY` and `ORDER BY` in a query

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

The two tables used throughout:

```
MEMBER(MemberID, Name, JoinDate, Active)
LOAN(LoanID, MemberID, BookTitle, LoanDate, Fine)
```

`MemberID` is the primary key of `MEMBER` and a foreign key in `LOAN`.

Choosing the data type

A `CREATE TABLE` question gives a mark for the types as well as for the structure, and the type has to suit what the field actually holds.

Type	Use it for	The trap
`INTEGER`	a count, an ID, a whole number	not for a phone number or a postcode: leading zeros are lost and no arithmetic is ever done on them
`REAL`	a measurement, a price, an average	
`VARCHAR(n)`	text of varying length, up to n characters	the usual right answer for a name or a title
`CHARACTER` (`CHAR(n)`)	text of a **fixed** length, such as a two-letter code	wastes space on anything variable
`BOOLEAN`	a field with exactly two states – yes/no, true/false	`VARCHAR(3)` holding `'Yes'` earns nothing where a `BOOLEAN` is meant
`DATE`	a calendar date	never `VARCHAR`: a text date cannot be compared or sorted correctly
`TIME`	a time of day	

Building the structure

```
CREATE DATABASE Library;

CREATE TABLE MEMBER (
  MemberID INTEGER PRIMARY KEY,
  Name VARCHAR(50),
  JoinDate DATE,
  Active BOOLEAN
);

CREATE TABLE LOAN (
  LoanID INTEGER PRIMARY KEY,
  MemberID INTEGER,
  BookTitle VARCHAR(100),
  LoanDate DATE,
  Fine REAL,
  FOREIGN KEY (MemberID) REFERENCES MEMBER(MemberID)
);
```

- The **foreign key** line names the field in **this** table, then the table and field it points at. Both halves are needed.
- To add a key to a table that already exists, use `ALTER TABLE`:

```
ALTER TABLE LOAN ADD FOREIGN KEY (MemberID) REFERENCES MEMBER(MemberID);
```

`ALTER TABLE` also adds a column: `ALTER TABLE MEMBER ADD Email VARCHAR(100);`

Querying two tables with INNER JOIN

```
SELECT C.Name, COUNT(D.DeviceID) AS Devices          ..... the fields to output; a calculated column, given a name
FROM CUSTOMER C                                      ..... the first table, with a short alias
INNER JOIN DEVICE D ON C.CustomerID = D.CustomerID   ..... the second table, linked through the foreign key
WHERE D.Type = 'phone'                               ..... keep only the rows that match
GROUP BY C.Name                                      ..... one output row per customer
ORDER BY Devices DESC;                               ..... sort the result, largest first
```

An `INNER JOIN` puts the two tables side by side and keeps only the rows whose keys match. The `ON` clause is always **foreign key = primary key**.

```
SELECT M.Name, L.BookTitle
FROM MEMBER M INNER JOIN LOAN L
  ON M.MemberID = L.MemberID
WHERE L.Fine > 0;
```

- An **alias** (`MEMBER M`) saves writing the table name in full, and it is needed wherever a field name appears in both tables – `MemberID` here. Write `M.MemberID`, not a bare `MemberID`.
- **Forgetting the `ON` clause is the error that costs most marks.** Without it the query pairs *every* member with *every* loan, so 100 members and 400 loans give 40,000 rows instead of 400.

Aggregates and GROUP BY

An **aggregate function** turns many rows into one value: `COUNT` how many, `SUM` the total, `AVG` the mean.

```
SELECT M.Name, COUNT(L.LoanID) AS NumLoans
FROM MEMBER M INNER JOIN LOAN L
  ON M.MemberID = L.MemberID
GROUP BY M.Name
ORDER BY NumLoans DESC;
```

- `GROUP BY` collapses the rows into one row per group. **Every field in the `SELECT` that is not inside an aggregate must appear in the `GROUP BY`** – here that is `M.Name`.
- Without `GROUP BY`, `COUNT` returns one number for the whole table: `SELECT COUNT(*) FROM LOAN;` is the total number of loans.
- The clauses must be written in this order, and marks are lost for writing them in another:

`SELECT` → `FROM` → `INNER JOIN ... ON` → `WHERE` → `GROUP BY` → `ORDER BY`

2 Practice

Now use the methods above.

2.1 Give an appropriate SQL data type for each of these fields. [4]

(a) `Active` – whether a member's card is still valid

(b) `JoinDate` – the date a member joined

(c) `Fine` – an amount of money owed, such as 2.50

(d) `BookTitle` – the title of a book

2.2 Write the SQL statement that creates a database called `Library`. [1]

2.3 Write the SQL statement that adds `MemberID` in `LOAN` as a foreign key referring to `MEMBER`, given that both tables already exist. [2]

2.4 Write an SQL query that gives the **total number of loans** in the `LOAN` table. [2]

2.5 State the order in which these clauses must be written in a query: `GROUP BY`, `SELECT`, `WHERE`, `FROM`, `ORDER BY`. [2]

3 Exam-style questions

3.1 Write SQL (DDL) to create the table `LOAN` with the fields `LoanID`, `MemberID`, `BookTitle`, `LoanDate` and `Fine`.

`LoanID` is the primary key and `MemberID` is a foreign key referring to `MEMBER`. Choose an appropriate data type for every field. [5]

3.2 Write an SQL query that lists the **name** of each member together with the **titles** of the books they currently have on loan. [4]

3.3 Write an SQL query that gives, for each member, their **name** and the **number of loans** they have, with the busiest member first. [4]

3.4 Write an SQL query that gives the **average fine** owed by members who joined before 1 January 2024. [3]

3.5 A student writes this query and is surprised that it returns 40,000 rows, when `MEMBER` holds 100 rows and `LOAN` holds 400.

```
SELECT M.Name, L.BookTitle
FROM MEMBER M INNER JOIN LOAN L;
```

Explain what is wrong and write the corrected query. [3]

Solutions

Each answer shows the steps, then the **result**.

2.1

(a) `BOOLEAN` – it has exactly two states

(b) `DATE`

(c) `REAL` – it has a fractional part (also accepted: `DECIMAL(6,2)`)

(d) `VARCHAR(100)` – text of varying length; any sensible n is accepted

2.2

- `CREATE DATABASE Library;`

2.3

- `ALTER TABLE LOAN`
- `ADD FOREIGN KEY (MemberID) REFERENCES MEMBER(MemberID);`

2.4

- `SELECT COUNT(*)` – or `COUNT(LoanID)`
- `FROM LOAN;`
- no `GROUP BY`, because the question wants one number for the whole table

2.5

- `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `ORDER BY`
- an `INNER JOIN ... ON` goes with the `FROM`, between `FROM` and `WHERE`

3.1

- `CREATE TABLE LOAN (` with the closing `);`
- `LoanID INTEGER PRIMARY KEY`
- `MemberID INTEGER`, `LoanDate DATE`
- `BookTitle VARCHAR(100)`, `Fine REAL`
- `FOREIGN KEY (MemberID) REFERENCES MEMBER(MemberID)`

```
CREATE TABLE LOAN (
  LoanID INTEGER PRIMARY KEY,
  MemberID INTEGER,
  BookTitle VARCHAR(100),
  LoanDate DATE,
  Fine REAL,
  FOREIGN KEY (MemberID) REFERENCES MEMBER(MemberID)
);
```

3.2

- `SELECT M.Name, L.BookTitle`

- `FROM MEMBER M INNER JOIN LOAN L`
- `ON M.MemberID = L.MemberID;`

```
SELECT M.Name, L.BookTitle
FROM MEMBER M INNER JOIN LOAN L
  ON M.MemberID = L.MemberID;
```

- a query that lists both tables without an `ON` earns the first mark only

3.3

- `SELECT M.Name, COUNT(L.LoanID)`
- the join with its `ON` clause
- `GROUP BY M.Name`
- `ORDER BY` the count `DESC`

```
SELECT M.Name, COUNT(L.LoanID) AS NumLoans
FROM MEMBER M INNER JOIN LOAN L
  ON M.MemberID = L.MemberID
GROUP BY M.Name
ORDER BY NumLoans DESC;
```

- `M.Name` is in the `SELECT` and is not inside an aggregate, so it **must** be in the `GROUP BY`

3.4

- `SELECT AVG(L.Fine)`
- the join with its `ON` clause
- `WHERE M.JoinDate < '2024-01-01';`

```
SELECT AVG(L.Fine)
FROM MEMBER M INNER JOIN LOAN L
  ON M.MemberID = L.MemberID
WHERE M.JoinDate < '2024-01-01';
```

3.5

- the `ON` clause is **missing**, so nothing says which rows of the two tables belong together
- every member is therefore paired with every loan, giving $100 \times 400 = 40\,000$ rows
- the corrected query adds `ON M.MemberID = L.MemberID;`, which keeps only the 400 pairings where the foreign key matches the primary key