A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

8.3 Data Definition Language (DDL) and Data Manipulation Language (DML)

Name: _____ Class: _____ Date: _____ **Total: 39 marks**

KEY WORDS data definition language 数据定义语言 • data manipulation language 数据操纵语言 • query 查询 • dbms 数据库管理系统 • join 连接

Objective

Build the skills to answer exam questions on **SQL** 结构化查询语言 — the difference between DDL and DML, and reading and writing simple SQL statements.

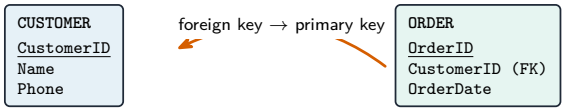
You must be able to:

- explain that the DBMS uses **DDL** to build the structure and **DML** to work with the data
- read and write simple **SQL (DDL)** statements (CREATE, ALTER, DROP)
- write **SQL (DML)** to query or change data in up to two tables

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ DDL or DML?



DDL builds the tables and keys that DML statements then query

SQL has two halves: **Data Definition Language (DDL)** changes the **structure** (tables, keys); **Data Manipulation Language (DML)** works with the **data** (query, insert, update, delete).

■ DDL — building the structure

```
CREATE TABLE Student (
  StudentID INTEGER PRIMARY KEY,
  Name      VARCHAR(50) NOT NULL,
  Year      INTEGER
);

ALTER TABLE Student ADD Email VARCHAR(100);
```

A **foreign key** is declared with `FOREIGN KEY … REFERENCES`:

```
CREATE TABLE Grade (
  StudentID INTEGER,
  Mark      INTEGER,
  FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
);
```

■ DML — working with the data

```
INSERT INTO Student (StudentID, Name, Year) VALUES (7, 'Lee', 12);

SELECT Name FROM Student WHERE Year = 12 ORDER BY Name;

UPDATE Student SET Year = 13 WHERE StudentID = 7;

DELETE FROM Student WHERE Year = 13;
```

To query **two tables**, match the foreign key to the primary key:

```
SELECT Student.Name, Grade.Mark
FROM   Student, Grade
WHERE  Student.StudentID = Grade.StudentID;
```

2 Practice

Now apply the methods above.

2.1 State the difference between **DDL** and **DML**. [2]

2.2 Write an SQL statement to list the **Name** of every student in **Year 12**. [2]

2.3 Write an SQL statement to **insert** a new student (StudentID 9, name "Sam", Year 11). [2]

2.4 Name the SQL statement used to **change a table's structure**. [1]

---

3 Exam-style questions

3.1 Write SQL **DDL** to create a table `Book` with: `BookID` (integer, primary key), `Title` (text, must not be empty), `Author` (text). [3]

3.2 Write SQL to change the `Year` of the student with `StudentID` 5 to 13. [2]

3.3 Tables `Student(StudentID, Name)` and `Grade(StudentID, Mark)` are linked by `StudentID`. Write SQL to list each student's **Name** with their **Mark**. [3]

3.4 Write SQL to **delete** every student in Year 13. [2]

3.5 Write SQL **DDL** to create a table `Grade` with `StudentID` (integer) as a **foreign key** referencing `Student(StudentID)`, and `Mark` (integer). [3]

3.6 Write SQL to list the **Name** of every Year-12 student, sorted into **alphabetical order** of name. [3]

3.7 A shipping company's database has two tables:

`SHIP(ShipID, ShipName, Capacity)` and `CONTAINER(ContainerID, ShipID, Contents, Weight)`

(a) **Describe** the relationship between the two tables, referring to the **primary** and **foreign** keys. [3]

(b) The company must record the date each container was last inspected. **Write** the SQL to add a suitable field to the `CONTAINER` table. [2]

(c) **Write** an SQL script to return the **number** of containers stored for the ship whose `ShipName` is `'Aurora'`. [4]

(d) **Write** an SQL script to list the `ShipName` and the total `Weight` carried, for every ship carrying more than 500 tonnes, heaviest first. [4]

(e) **Describe** the purpose of the **developer interface** in a DBMS, and **give** one feature it provides that the query processor does not. [3]

Solutions

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

2.1

- DDL defines/changes the **structure** (tables, keys)
- DML works with the **data** (insert, update, delete, query)

2.2

- `SELECT … FROM` the student table, then filter Year 12
- `SELECT Name FROM Student WHERE Year = 12;`

2.3

- `INSERT INTO … VALUES`
- `INSERT INTO Student (StudentID, Name, Year) VALUES (9, 'Sam', 11);`

2.4

- `ALTER` (TABLE) — a DDL statement

3.1

```
CREATE TABLE Book (
  BookID INTEGER PRIMARY KEY,
  Title  VARCHAR(100) NOT NULL,
  Author VARCHAR(50)
);
```

- `CREATE TABLE` + fields; `PRIMARY KEY`; `NOT NULL` on Title

3.2

- `UPDATE … SET`
- `UPDATE Student SET Year = 13 WHERE StudentID = 5;`

3.3

- both tables, joined on `StudentID`
- `SELECT Student.Name, Grade.Mark FROM Student, Grade WHERE Student.StudentID = Grade.StudentID;`

3.4

- `DELETE FROM`
- `DELETE FROM Student WHERE Year = 13;`

3.5


```
CREATE TABLE Grade (
  StudentID INTEGER,
  Mark      INTEGER,
  FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
);
```

- `CREATE TABLE` + fields; `FOREIGN KEY`; `REFERENCES` the correct table/field

3.6

- `SELECT … WHERE` and `ORDER BY`
- `SELECT Name FROM Student WHERE Year = 12 ORDER BY Name;`

3.7

(a) one ship has **many** containers, so it is a one-to-many relationship

- `ShipID` is the **primary key** of `SHIP`, uniquely identifying each ship
- `ShipID` in `CONTAINER` is a **foreign key** referring to it, which links each container to its ship

(b) `ALTER TABLE CONTAINER`

- `ADD LastInspected DATE;`

(c)

```
SELECT COUNT(*)
FROM CONTAINER, SHIP
WHERE CONTAINER.ShipID = SHIP.ShipID
  AND SHIP.ShipName = 'Aurora';
```

- `SELECT COUNT`; both tables; the join condition; the name condition, quoted

(d)

```
SELECT ShipName, SUM(Weight)
FROM SHIP, CONTAINER
WHERE SHIP.ShipID = CONTAINER.ShipID
GROUP BY ShipName
HAVING SUM(Weight) > 500
ORDER BY SUM(Weight) DESC;
```

- `SUM` with `GROUP BY`; the join; `HAVING` — not `WHERE`, because the condition is on an aggregate; `ORDER BY … DESC`

(e) the developer interface is the part of the DBMS used to **create and maintain** the database

- defining tables and data types, setting keys and relationships, and managing users and access rights
- the query processor only **runs** queries against a structure that already exists — it cannot create it