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

Name: _____ Class: _____ Date: _____ **Total: 46 marks**

KEY WORDS data definition language • data manipulation language • query • dbms • join

Objective

Build the skills to answer exam questions on **SQL** – the difference between DDL and DML, and reading and writing simple SQL statements.

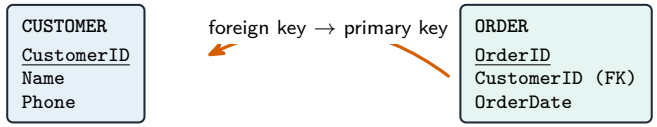
You must be able to:

- 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 Key knowledge and 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;
```

■ Key knowledge: SQL conditions and keys

`LIKE 'Li%'` matches text starting with `Li`; `%` represents any number of characters and `_` exactly one. `AND` requires both conditions; `OR` requires at least one. Use brackets when combining them.

```
CREATE DATABASE Club;
CREATE TABLE Booking (
  MemberID INTEGER,
  EventID INTEGER,
  PRIMARY KEY (MemberID, EventID)
);
ALTER TABLE Booking ADD FOREIGN KEY (MemberID)
  REFERENCES Member(MemberID);
SELECT Name FROM Member
WHERE Name LIKE 'Li%' AND (Year = 12 OR Year = 13)
ORDER BY Name DESC;
```

The composite primary key makes the **pair** unique, not each column separately. A foreign key prevents a non-null reference to a missing parent record. `ON UPDATE CASCADE` carries a parent-key change to referencing rows; `ON DELETE CASCADE` deletes those rows when the parent is deleted.

■ Connect the design to the SQL

The primary and foreign keys designed in sheets 8.1-8.1.2 become constraints in `CREATE TABLE`. A composite primary key is declared once with both fields: `PRIMARY KEY (MemberID, EventID)`. Declaring each field separately as a primary key would be wrong.

Read `FOREIGN KEY (StudentID) REFERENCES Student(StudentID)` in two parts: the field in this table, then the table and key it must match. Create the referenced table before the referencing table. Use compatible data types for the two fields.

For the Student rows `(7, Lee, 12)`, `(8, Lin, 13)` and `(9, Mei, 12)`, the worked `SELECT` returns Lee and Mei in that order. `WHERE` selects the records; `SELECT Name` chooses the displayed field; `ORDER BY Name` orders the result. The UPDATE and DELETE examples change the stored data, whereas SELECT returns a result.

2 Practice

Now apply the methods above.

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

DDL | DML

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

Student

StudentID	Name	Year
3	Ana	12
5	Ben	13
7	Chen	12
8	Dev	13

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

Student

StudentID	Name	Year
3	Ana	12
5	Ben	13
7	Chen	12
8	Dev	13

9	Sam	11

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]

Book

BookID	Title	Author

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

Student

StudentID	Name	Year
3	Ana	12
5	Ben	13
7	Chen	12
8	Dev	13

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]

Student

StudentID	Name
3	Ana
5	Ben
7	Chen

Grade

StudentID	Mark
3	71
5	58
7	84

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

Student

StudentID	Name	Year
3	Ana	12
5	Ben	13
7	Chen	12
8	Dev	13

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

Student

StudentID	Name

Grade

StudentID	Mark

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

Student

StudentID	Name	Year
3	Ana	12
5	Ben	13
7	Chen	12
8	Dev	13

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]

ship and its containers

(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]

3.8 Table `Member(MemberID, Name, Year)` holds club members. Write SQL to select the names starting with `An` whose year is either 12 or 13, in descending name order. Then define `Attendance(MemberID, EventID)` with a composite primary key and a foreign key referencing `Member`. [7]

`Member`

MemberID	Name	Year
101	Anna	12
102	Ben	13
103	Andre	11
104	Ana	13

`Attendance`

MemberID	EventID