Exercise walk-through

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

eXercise

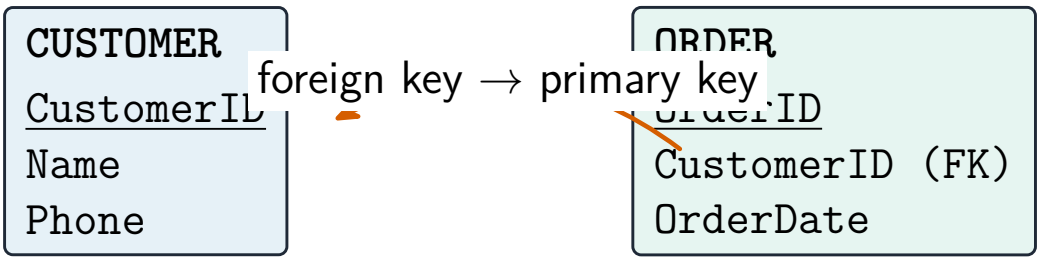
You must be able to

- 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

Method — DDL or DML?

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).

Method — DDL or DML?



DDL builds the tables and keys that DML statements then query

Method — DDL — building the structure

```
kCREATE+w kTABLE+w nStudent+w p(
+w  nStudentID+w n+nbINTEGER+w kPRIMARY+w kKEYp,
+w  nName+w      n+nbVARCHARp(1+m+mi50p)+w kNOT+w kNULLp,
+w  kYear+w      n+nbINTEGER
p)p;
```

```
kALTER+w kTABLE+w nStudent+w kADD+w nEmail+w n+nbVARCHARp(1+m+mi100p)p;
```

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

```
kCREATE+w kTABLE+w nGrade+w p(
+w  nStudentID+w n+nbINTEGERp,
+w  nMark+w      n+nbINTEGERp,
+w  kFOREIGN+w kKEY+w p(nStudentIDp)+w kREFERENCES+w nStudentp(nStudentIDp)
p)p;
```

Method — DML — working with the data

```
kINSERT+w kINTO+w nStudent+w p(nStudentIDp,+w nNamep,+w kYearp)+w kVALUES+w p(l+m+mi7p,+w err‘nLeeerr
```

```
kSELECT+w nName+w kFROM+w nStudent+w kWHERE+w kYear+w o=+w l+m+mi12+w kORDER+w kBY+w nNamep;
```

```
kUPDATE+w nStudent+w kSET+w kYear+w o=+w l+m+mi13+w kWHERE+w nStudentID+w o=+w l+m+mi7p;
```

```
kDELETE+w kFROM+w nStudent+w kWHERE+w kYear+w o=+w l+m+mi13p;
```

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

```
kSELECT+w nStudentp.nNamep,+w nGradep.nMark
kFROM+w   nStudentp,+w nGrade
kWHERE+w  nStudentp.nStudentID+w o=+w nGradep.nStudentIDp;
```

Practice 2.1 ▪ 2 marks

State the difference between **DDL** and **DML**.

Solution 2.1

Worked steps

DDL defines/changes the **structure** (tables, keys); DML works with the **data** (insert, update, delete, query) B1 B1.

Practice 2.2 ▪ 2 marks

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

Solution 2.2

Worked steps

`SELECT Name FROM Student WHERE Year = 12;` **M1** **A1** *SELECT…FROM, correct WHERE.*

Practice 2.3 ▪ 2 marks

Write an SQL statement to **insert** a new student (StudentID 9, name “Sam”, Year 11).

Solution 2.3

Method: DML — working with the data

Worked steps

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

M1 A1 *INSERT…VALUES, correct values.*

Practice 2.4 ▪ 1 mark

Name the SQL statement used to **change a table's structure**.

Solution 2.4

Method: DDL — building the structure

Worked steps

`ALTER` (TABLE) — a DDL statement B1.

Exam-style 3.1 ▪ 3 marks

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

Solution 3.1

> Method: DDL — building the structure

Worked steps

```
kCREATE+w kTABLE+w nBook+w p(
+w  nBookID+w n+nbINTEGER+w kPRIMARY+w kKEYp,
+w  nTitle+w  n+nbVARCHARp(l+m+mi100p)+w kNOT+w kNULLp,
+w  nAuthor+w n+nbVARCHARp(l+m+mi50p)
p)p;
```

Solution 3.1

M1 A1 A1 *CREATE TABLE + fields, PRIMARY KEY, NOT NULL on Title*

Exam-style 3.2 ▪ 2 marks

Write SQL to change the `Year` of the student with `StudentID` 5 to 13.

Solution 3.2

Worked steps

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

M1 A1 *UPDATE…SET, correct WHERE.*

Exam-style 3.3 ▪ 3 marks

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

Solution 3.3

Method: DML — working with the data

Worked steps

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

M1 M1 A1 *both tables, join condition, correct fields.*

Exam-style 3.4 ▪ 2 marks

Write SQL to **delete** every student in Year 13.

Solution 3.4

Method: DML — working with the data

Worked steps

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

M1 A1 *DELETE FROM, correct WHERE.*

Exam-style 3.5 ▪ 3 marks

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

Solution 3.5

> Method: DDL — building the structure

Worked steps

```
kCREATE+w kTABLE+w nGrade+w p(
+w  nStudentID+w n+nbINTEGERp,
+w  nMark+w      n+nbINTEGERp,
+w  kFOREIGN+w kKEY+w p(nStudentIDp)+w kREFERENCES+w nStudentp(nStudentIDp)
p)p;
```

Solution 3.5

M1 M1 A1 *CREATE TABLE + fields, FOREIGN KEY, REFERENCES the correct table/field*

Exam-style 3.6 ▪ 3 marks

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

Solution 3.6

Method: DML — working with the data

Worked steps

```
SELECT Name FROM Student WHERE Year = 12 ORDER BY Name;
```
M1 M1 A1

SELECT…WHERE, ORDER BY, fully correct.

Exam-style 3.7 ▪ 3 marks

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.
(b) The company must record the date each container was last inspected. **Write** the SQL to add a suitable field to the `CONTAINER` table.
(c) **Write** an SQL script to return the **number** of containers stored for the ship whose `ShipName` is `'Aurora'`.
(d) **Write** an SQL script to list the `ShipName` and the total `Weight` carried, for every ship carrying more than 500 tonnes, heaviest first.
(e) **Describe** the purpose of the **developer interface** in a DBMS, and **give** one feature it provides that the query processor does not.

Solution 3.7

Method: DML — working with the data

Worked steps

(a) One ship has **many** containers, so it is a one-to-many relationship B1. `ShipID` is the **primary key** of `SHIP`, uniquely identifying each ship B1; `ShipID` in `CONTAINER` is a **foreign key** referring to it, which is what links each container to its ship B1.
(b) `ALTER TABLE CONTAINER` B1 `ADD LastInspected DATE;` B1.
(c)

```
kSELECT+w kCOUNTp(o*p)
kFROM+w nCONTAINERp,+w nSHIP
kWHERE+w nCONTAINERp.nShipID+w o=+w nSHIPp.nShipID
+w  kAND+w nSHIPp.nShipName+w o=+w err‘nAuroraerrp;
```

Solution 3.7

B1 B1 B1 B1 *'SELECT COUNT'; both tables; the join condition; the name condition, quoted*

(d)

```
kSELECT+w nShipNamep,+w kSUMp(nWeightp)
kFROM+w nSHIPp,+w nCONTAINER
kWHERE+w nSHIPp.nShipID+w o=+w nCONTAINERp.nShipID
kGROUP+w kBY+w nShipName
kHAVING+w kSUMp(nWeightp)+w o+w l+m+mi500
kORDER+w kBY+w kSUMp(nWeightp)+w kDESCp;
```

Solution 3.7

B1 B1 B1 B1 *'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 B1: defining tables and data types, setting keys and relationships, and managing users and access rights B1. The query processor only **runs** queries against a structure that already exists — it cannot create it B1.