

Session 10

Mapping classes to database tables

Object to relational mapping follows fixed rules: a class becomes a table, an object a row, a one-to-many association a foreign key on the many side, and a many-to-many association (or an association class) a junction table.

Objectives

✗ Do not copy. Read for understanding and the viva

- Complete questions 23 to 23 of the manual: mapping classes to database tables
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✗ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q23	Do mapping of the following Classes into database tables	Complete

Preparation

✗ Do not copy. Read for understanding and the viva

- The manual asks for a problem description of 300 to 500 words and a list of assumptions before every diagram. Write both first; they fix the scope the evaluator marks you against.
- Write the mapping rule you apply next to each table: class to table, attribute to column, association to foreign key or junction table.
- Give every table a primary key, and write the `CREATE TABLE` statements with foreign key constraints so the mapping is checkable.

Question 23

Problem Statement

■ Write in lab record

Do mapping of the following Classes into database tables

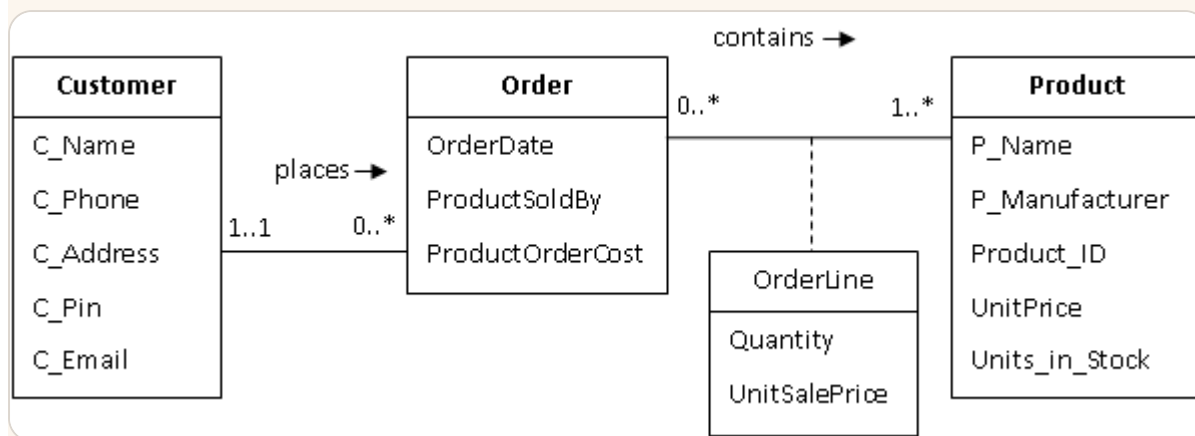


Figure 1.18: Customer Order Association Class

Solution

■ Write in lab record

Assumptions

The shop of Session 8, Question 20 now needs its data in a relational database so that orders survive after the program exits and can be queried by report tools. The figure has three classes and one association class. A customer places any number of orders and every order belongs to exactly one customer. An order contains one or more products, a product may appear in many orders, and the quantity and unit sale price of each (order, product) pair are stored on the OrderLine association class. The database must keep the same facts with the same names, enforce that an order cannot refer to a customer who does not exist, that an order line cannot refer to a missing order or product, that the same product is not listed twice on one order, and that a quantity is positive. It must also support one query that walks the whole figure from customer to product.

- Target is MySQL 8. Types are chosen because the figure gives none: `VARCHAR` for text, `CHAR(6)` for the PIN, `DECIMAL(10, 2)` for money, `DATE` for the order date, `INT` for counts and IDs.

- Customer and Order have no identifying attribute in the figure, so each gets a surrogate `AUTO_INCREMENT` key. Product already has `Product_ID`, which becomes its primary key.
- `Order` is a reserved word in SQL, so the table is `Orders`; column names keep the figure's spelling.
- `C_Email` is unique per customer. `ProductOrderCost` is kept as a column because the figure shows it as an attribute, even though it can be derived from the lines.
- Deleting an order deletes its lines (`ON DELETE CASCADE`); deleting a product or customer that is still referenced is refused (the default `RESTRICT`).

Mapping rules

Rule	Applies to	Result
1. Class to table, attribute to column, one primary key per table	Customer, Product, Order	Customer(customer_id PK, C_Name, C_Phone, C_Address, C_Pin, C_Email) ; Product(Product_ID PK, P_Name, P_Manufacturer, UnitPrice, Units_in_Stock) ; Orders(order_id PK, OrderDate, ProductSoldBy, ProductOrderCost)
2. One-to-many association: foreign key on the many side	places, Customer 1..1 to 0..* Order	Orders.customer_id references Customer.customer_id , NOT NULL because the lower bound at Customer is 1
3. Many-to-many association: junction table keyed by both foreign keys	contains, Order 0..* to 1..* Product	OrderLine(order_id, Product_ID) with primary key on the pair
4. Association class: its attributes become columns of the junction table	OrderLine with Quantity, UnitSalePrice	OrderLine.Quantity , OrderLine.UnitSalePrice
5. Multiplicity lower bounds become NOT NULL or CHECK constraints	1..1 at Customer, positive quantity	customer_id NOT NULL , CHECK (Quantity > 0)
6. Object identity that the figure does not name becomes a surrogate key	Customer, Order	customer_id , order_id with AUTO_INCREMENT

Resulting schema, one line per table (PK underlined in a hand drawing, FK marked):

```

1 Customer  (customer_id PK, C_Name, C_Phone, C_Address, C_Pin, C_Email)
2 Product   (Product_ID PK, P_Name, P_Manufacturer, UnitPrice, Units_in_Stock)
3 Orders    (order_id PK, OrderDate, ProductSoldBy, ProductOrderCost, customer
4 OrderLine (order_id FK → Orders, Product_ID FK → Product, Quantity, UnitSa
5           PK = (order_id, Product_ID)

```

Foreign keys drawn as arrows from the referencing column to the referenced key:

```

1  +-----+          +-----+          +-----+
2  | Customer  |          | Orders      |          | Product
3  |-----|          |-----|          |-----|
4  | customer_id |<-----| customer_id (FK) |          | Product_ID (PK)
5  | C_Name     | 1..1 | order_id   (PK) |          | P_Name
6  | C_Phone    |          | OrderDate          |          | P_Manufacturer
7  | C_Address  |          | ProductSoldBy       |          | UnitPrice
8  | C_Pin      |          | ProductOrderCost    |          | Units_in_Stock
9  | C_Email    |          +-----+          +-----+
10 +-----+          ^                      ^
11          | 0..*          | 1..*
12          +-----+-----+-----+-----+
13          | OrderLine |
14          |-----|
15          | order_id (FK, PK part)  Product_ID (FK, PK part)|
16          | Quantity                UnitSalePrice          |
17          +-----+-----+-----+-----+

```

The same four tables as PlantUML, for a rendered copy of the schema:

customer_order_tables.puml

```
1 @startuml
2 hide circle
3 skinparam linetype ortho
4
5 entity "Customer" as customer {
6     * customer_id : INT <<PK>>
7     --
8     C_Name : VARCHAR(100)
9     C_Phone : VARCHAR(15)
10    C_Address : VARCHAR(200)
11    C_Pin : CHAR(6)
12    C_Email : VARCHAR(100)
13 }
14
15 entity "Orders" as orders {
16     * order_id : INT <<PK>>
17     --
18     OrderDate : DATE
19     ProductSoldBy : VARCHAR(100)
20     ProductOrderCost : DECIMAL(10,2)
21     customer_id : INT <<FK>>
22 }
23
24 entity "Product" as product {
25     * Product_ID : INT <<PK>>
26     --
27     P_Name : VARCHAR(100)
28     P_Manufacturer : VARCHAR(100)
29     UnitPrice : DECIMAL(10,2)
30     Units_in_Stock : INT
31 }
32
33 entity "OrderLine" as orderline {
34     * order_id : INT <<PK, FK>>
35     * Product_ID : INT <<PK, FK>>
36     --
37     Quantity : INT
38     UnitSalePrice : DECIMAL(10,2)
39 }
40
41 customer ||--o{ orders : places
42 orders ||--|{ orderline : has lines
```

```

43 product |←-o{ orderline : appears in
44 @enduml

```

How the objects of Session 8, Question 20 become rows. One object is one row; one link is one foreign key value; one OrderLine object is one row of the junction table:

Object in the C++ program	Table	Row
asha (Customer)	Customer	1, Asha, 9876543210, 12 MG Road Jaipur, 302001, asha@example.com
pen, notebook, stapler (Product)	Product	101, Pen, Cello, 10.00, 500 and two more rows
o1 placed by asha	Orders	1, 2026-09-01, Store counter, 415.00, customer_id = 1
o2 placed by asha	Orders	2, 2026-09-15, OnLine, 230.00, customer_id = 1
OrderLine o1 to pen, 20 at 9.50	OrderLine	1, 101, 20, 9.50
OrderLine o1 to notebook, 5 at 45.00	OrderLine	1, 102, 5, 45.00
OrderLine o2 to stapler, 2 at 115.00	OrderLine	2, 103, 2, 115.00
pointer o1.customer	Orders.customer_id	the value 1, not a separate row
vector asha.orders	none	recovered by <code>SELECT ... FROM Orders WHERE customer_id = 1</code> ; the database stores the link once, on the many side

Steps

1. Save the listing below as `customer_order.sql` in a `session-10` folder.

2. Start MySQL and run `mysql -u root -p < customer_order.sql` . The script creates the database, the four tables, the sample rows and runs the join.
3. To see the tables in the MySQL shell: `USE customer_order; SHOW TABLES; DESCRIBE OrderLine;` .
4. Paste the result of the final `SELECT` into the record.

SQL

customer_order.sql

```
1  -- customer_order.sql -- MCSL-222 Session 10, Q23
2  -- Figure 1.18 (Customer, Order, Product, OrderLine) mapped to MySQL tables.
3  -- Run: mysql -u root -p < customer_order.sql
4
5  DROP DATABASE IF EXISTS customer_order;
6  CREATE DATABASE customer_order;
7  USE customer_order;
8
9  -- Rule 1: class → table, attribute → column, one surrogate primary key pe
10 CREATE TABLE Customer (
11     customer_id INT AUTO_INCREMENT PRIMARY KEY,
12     C_Name      VARCHAR(100) NOT NULL,
13     C_Phone     VARCHAR(15),
14     C_Address   VARCHAR(200),
15     C_Pin       CHAR(6),
16     C_Email     VARCHAR(100) UNIQUE
17 );
18
19 -- Product_ID already identifies a product in the figure, so it is the key.
20 CREATE TABLE Product (
21     Product_ID INT PRIMARY KEY,
22     P_Name      VARCHAR(100) NOT NULL,
23     P_Manufacturer VARCHAR(100),
24     UnitPrice   DECIMAL(10, 2) NOT NULL,
25     Units_in_Stock INT NOT NULL DEFAULT 0
26 );
27
28 -- Rule 2: one-to-many (Customer 1..1 places 0..* Order) → foreign key on t
29 -- many side. NOT NULL because the multiplicity at Customer is exactly 1.
30 -- "Order" is a reserved word in SQL, so the table is named Orders.
31 CREATE TABLE Orders (
32     order_id INT AUTO_INCREMENT PRIMARY KEY,
33     OrderDate DATE NOT NULL,
34     ProductSoldBy VARCHAR(100),
35     ProductOrderCost DECIMAL(10, 2),
36     customer_id INT NOT NULL,
37     FOREIGN KEY (customer_id) REFERENCES Customer (customer_id)
38 );
39
40 -- Rule 3: many-to-many with an association class (Order 0..* contains 1..*
41 -- Product, OrderLine) → junction table whose primary key is the pair of
42 -- foreign keys, plus the association-class attributes as columns.
43 CREATE TABLE OrderLine (
```

```
44     order_id      INT NOT NULL,
45     Product_ID    INT NOT NULL,
46     Quantity      INT NOT NULL CHECK (Quantity > 0),
47     UnitSalePrice DECIMAL(10, 2) NOT NULL,
48     PRIMARY KEY (order_id, Product_ID),
49     FOREIGN KEY (order_id) REFERENCES Orders (order_id) ON DELETE CASCADE,
50     FOREIGN KEY (Product_ID) REFERENCES Product (Product_ID)
51 );
52
53 -- Sample rows (same data as the C++ program of Session 8, Q20)
54 INSERT INTO Customer (C_Name, C_Phone, C_Address, C_Pin, C_Email) VALUES
55     ('Asha', '9876543210', '12 MG Road, Jaipur', '302001', 'asha@example.com
56     ('Ravi', '9123456780', '4 FC Road, Pune', '411004', 'ravi@example.com
57
58 INSERT INTO Product (Product_ID, P_Name, P_Manufacturer, UnitPrice, Units_in
59     (101, 'Pen', 'Cello', 10.00, 500),
60     (102, 'Notebook', 'Classmate', 45.00, 40),
61     (103, 'Stapler', 'Kangaro', 120.00, 3);
62
63 INSERT INTO Orders (OrderDate, ProductSoldBy, ProductOrderCost, customer_id)
64     ('2026-09-01', 'Store counter', 415.00, 1),
65     ('2026-09-15', 'Online', 230.00, 1),
66     ('2026-09-20', 'Online', 90.00, 2);
67
68 INSERT INTO OrderLine (order_id, Product_ID, Quantity, UnitSalePrice) VALUES
69     (1, 101, 20, 9.50),
70     (1, 102, 5, 45.00),
71     (2, 103, 2, 115.00),
72     (3, 102, 2, 45.00);
73
74 -- Walk the whole figure: Customer → Orders → OrderLine → Product
75 SELECT c.C_Name,
76        o.order_id,
77        o.OrderDate,
78        p.P_Name,
79        ol.Quantity,
80        ol.UnitSalePrice,
81        ol.Quantity * ol.UnitSalePrice AS line_total
82 FROM Customer c
83 JOIN Orders o ON o.customer_id = c.customer_id
84 JOIN OrderLine ol ON ol.order_id = o.order_id
85 JOIN Product p ON p.Product_ID = ol.Product_ID
86 ORDER BY o.order_id, p.Product_ID;
87
```

```

88 -- Check that the stored ProductOrderCost agrees with the sum of its lines
89 SELECT o.order_id,
90        o.ProductOrderCost,
91        SUM(ol.Quantity * ol.UnitSalePrice) AS computed_cost
92 FROM Orders o
93 JOIN OrderLine ol ON ol.order_id = o.order_id
94 GROUP BY o.order_id, o.ProductOrderCost
95 ORDER BY o.order_id;

```

Output

No MySQL server is installed on the machine used to prepare this page, so the script was run through SQLite 3 with `PRAGMA foreign_keys = ON` after replacing `INT AUTO_INCREMENT PRIMARY KEY` by `INTEGER PRIMARY KEY` and dropping the `CREATE DATABASE` and `USE` lines. Every `CREATE TABLE`, `INSERT` and the join ran without error and produced these rows; MySQL prints the same rows with `9.50` and `45.00` formatting because the columns are `DECIMAL`.

	C_Name	order_id	OrderDate	P_Name	Quantity	UnitSalePrice	line_total
1	-----	-----	-----	-----	-----	-----	-----
2							
3	Asha	1	2026-09-01	Pen	20	9.5	190.0
4	Asha	1	2026-09-01	Notebook	5	45	225
5	Asha	2	2026-09-15	Stapler	2	115	230
6	Ravi	3	2026-09-20	Notebook	2	45	90

Two statements were then run on purpose to show the constraints working:

```

1 INSERT INTO Orders (OrderDate, customer_id) VALUES ('2026-09-21', 99);
2 Error: FOREIGN KEY constraint failed
3
4 INSERT INTO OrderLine VALUES (1, 101, 3, 9.50);
5 Error: UNIQUE constraint failed: OrderLine.order_id, OrderLine.Product_ID

```

The first fails because customer 99 does not exist (rule 2). The second fails because order 1 already has a line for product 101 (rule 3: the pair is the primary key).

The last query in the script checks the stored `ProductOrderCost` against the sum of its lines; all three orders agree:

1	order_id	ProductOrderCost	computed_cost
2	-----	-----	-----
3	1	415	415.0
4	2	230	230
5	3	90	90

Explanation

Diagram element	Where it is in the SQL
Class Customer with five attributes	<code>CREATE TABLE Customer</code> with five columns plus the surrogate <code>customer_id</code> .
Class Product	<code>CREATE TABLE Product</code> ; <code>Product_ID INT PRIMARY KEY</code> because the figure already names an identifier.
Class Order	<code>CREATE TABLE Orders</code> (renamed to avoid the reserved word) with <code>OrderDate</code> , <code>ProductSoldBy</code> , <code>ProductOrderCost</code> .
places, 1..1 to 0..*	<code>customer_id INT NOT NULL</code> plus <code>FOREIGN KEY (customer_id) REFERENCES Customer (customer_id) in Orders</code> . The many side carries the key; <code>NOT NULL</code> is the <code>1..1</code> .
contains, 0..* to 1..*	<code>CREATE TABLE OrderLine</code> with two foreign keys, one to <code>Orders</code> and one to <code>Product</code> .
Association class OrderLine	The same junction table carries <code>Quantity</code> and <code>UnitSalePrice</code> . <code>PRIMARY KEY (order_id, Product_ID)</code> means one line per pair, exactly one object per link in the figure.
Lower bound 1..* at Product	Cannot be written as a table constraint in MySQL; the application inserts at least one line per order. State this in the viva when asked.
Whole figure in one query	The <code>SELECT</code> joins <code>Customer</code> to <code>Orders</code> on <code>customer_id</code> , <code>Orders</code> to <code>OrderLine</code> on <code>order_id</code> , and <code>OrderLine</code> to <code>Product</code> on <code>Product_ID</code> ; <code>Quantity * UnitSalePrice</code> is the line total.
Derived attribute ProductOrderCost	Stored as a column because the figure lists it; the <code>GROUP BY</code> query at the end of the script recomputes it from <code>OrderLine</code> so the two can be compared.

Viva Questions

✕ Do not copy. Read for understanding and the viva

Q: Which side of a one-to-many association gets the foreign key, and why? **A:** The many side. A row can hold one foreign key value, so the row that belongs to exactly one parent (`Orders`) stores the parent's key. Storing order ids in `Customer` would need a list in one column.

Q: How is a many-to-many association mapped? **A:** As a junction table with one foreign key to each side and a primary key on the pair. `OrderLine` is that table.

Q: Where do the attributes of an association class go? **A:** Into the junction table, because they belong to the link, not to either end. `Quantity` and `UnitSalePrice` are columns of `OrderLine`.

Q: Why does Product keep `Product_ID` as its key while Customer gets a new `customer_id`? **A:** The figure names an identifier for Product only. The other classes rely on object identity, which the database has to replace with a surrogate key.

Q: What does `NOT NULL` on `Orders.customer_id` correspond to in the figure? **A:** The lower bound 1 of `1..1` at Customer: every order must have a customer.

Q: Why `DECIMAL(10, 2)` and not `FLOAT` for prices? **A:** `DECIMAL` stores exact rupees and paise; `FLOAT` rounds, so totals drift.

Q: What happens when a customer with orders is deleted? **A:** The default `RESTRICT` on the foreign key refuses the delete. Deleting an order removes its lines through `ON DELETE CASCADE`.

Q: Can the database enforce that an order has at least one product? **A:** Not with a plain constraint, because the order row is inserted before any line. It is checked by the application or a trigger.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Naming the table `Order`; it is a reserved word and the `CREATE TABLE` fails.
- Putting the foreign key on the one side (an `order_id` column in `Customer`), which allows only one order per customer.
- Giving `OrderLine` its own `line_id` key and forgetting the unique pair, so the same product can be listed twice on one order.
- Leaving `customer_id` nullable, which drops the `1..1` multiplicity.
- Writing the `CREATE TABLE` statements in the wrong order, so a foreign key refers to a table that does not exist yet; parents first, `OrderLine` last.

- Forgetting the `INSERT` rows and the join query; the mapping is only shown to work when a query crosses every foreign key.

Session Summary

■ Write in lab record

- Question 23: mapping-rule table for figure 1.18 and `customer_order.sql` with `CREATE TABLE` for Customer, Product, Orders and OrderLine, primary and foreign keys, sample rows, and a four-table join with its output
- Problem description and assumptions for the figure, plus the constraint checks that fail on purpose

[✎ Edit this page](#)

[< Previous](#)
Session 9

Section 2 · Web Technologies [Next >](#)