

Database Design Report: Bookworm Books

Sample student report (model answer) | Module: [code and title] | Student: [name and ID] | Date: [date]

1. Introduction

This report designs a database for **Bookworm Books**, a small online bookshop. The scenario given was:

Bookworm Books is a small online bookshop. Customers register with their name and email address. A customer can place many orders. Each order has a date and can include several books. Every book has a title and a price. For each book on an order, we record how many copies were bought.

The report follows two routes to the same goal. **Part A** analyses the scenario and draws an Entity Relationship Diagram (ERD). **Part B** takes the data from the shop's paper order forms and normalises it from Unnormalised Form (UNF) to Third Normal Form (3NF). The two results are then compared as a check on the design.

2. Part A: Entity Relationship Diagram

2.1 Identifying the entities

I read the scenario and underlined the nouns. An entity is a thing the system must keep a list of, so I kept only the nouns where there will be many instances.

Candidate noun	Entity?	Reason
Customer	Yes	Many customers will register, and we store data about each.
Order	Yes	Many orders will be placed, and each has its own details.
Book	Yes	The shop sells many different books.
Bookworm Books	No	This is the business itself. There is only one, so there is nothing to list.
Name, email, date, title, price	No	These are facts about an entity, so they are attributes.
Copies bought (quantity)	No	This is a fact about a book <i>on an order</i> . It is dealt with in section 2.4.

Entities identified: CUSTOMER, ORDER, BOOK.

2.2 Identifying the attributes

An attribute is one fact about an entity. Each entity needs a primary key (PK), a value that is different for every row. None of the facts in the scenario is guaranteed to be unique (two customers could share a name, and two books could share a title), so I added an ID to each entity.

Entity	Attribute	Notes
CUSTOMER	CustomerID (PK)	Added ID, because names are not unique.
	CustomerName	Given in the scenario.
	CustomerEmail	Given in the scenario.
ORDER	OrderID (PK)	Added ID.
	OrderDate	Given in the scenario.
BOOK	BookID (PK)	Added ID.
	Title	Given in the scenario.
	Price	Given in the scenario.

2.3 Identifying the relationships

I looked for the verbs in the scenario and tested each link by reading it in both directions.

Relationship	Reading one way	Reading the other way	Cardinality
CUSTOMER places ORDER	One customer places many orders.	Each order is placed by one customer.	One-to-many (1:M)
ORDER contains BOOK	One order contains many books.	One book can appear on many orders.	Many-to-many (M:N)

2.4 Resolving the many-to-many relationship

A relational database cannot store a many-to-many relationship directly. I resolved it by adding a **linking entity**, ORDER_ITEM, between ORDER and BOOK. Each row of ORDER_ITEM represents one line on an order form: one order, one book, and a number of copies.

- ORDER_ITEM holds **OrderID** and **BookID** as foreign keys (FK). Together they form its composite primary key.
- **Quantity** is a fact about the *pair* (this book on this order), so it belongs in ORDER_ITEM. This answers the open question from section 2.1.
- The relationship is now two one-to-many relationships: ORDER to ORDER_ITEM, and BOOK to ORDER_ITEM.

I also applied the rule that the **PK from the "one" side is copied into the "many" side as an FK**. This is why ORDER gains **CustomerID (FK)**.

2.5 The ERD

Figure 1: ERD for Bookworm Books (crow's foot notation)

[Insert your drawn ERD here. Draw it in draw.io, Lucidchart or a similar tool.]

The diagram should show four boxes, left to right, with the attributes listed and PK and FK marked:

```
CUSTOMER ||-----o{ ORDER ||-----o{ ORDER_ITEM }o-----|| BOOK
(1)           places (many) has (many) is in (1)
```

Entity	Attributes
CUSTOMER	CustomerID (PK), CustomerName, CustomerEmail
ORDER	OrderID (PK), OrderDate, CustomerID (FK)
ORDER_ITEM	OrderID (PK, FK), BookID (PK, FK), Quantity
BOOK	BookID (PK), Title, Price

3. Part B: Normalisation

3.1 Why normalise?

If all the data is kept in one table, three problems arise:

- **Repeated data:** a customer's name and email are typed again on every order, so a change made in one place leaves the other copies out of date.
- **Insert problem:** a new book cannot be stored until somebody orders it.
- **Delete problem:** deleting a customer's only order also deletes the customer's details.

Normalisation avoids these problems by making sure **each fact is stored once**.

3.2 Step 1: Unnormalised Form (UNF)

The data below comes from three paper order forms typed into a single table.

OrderID	OrderDate	CustomerID	CustomerName	CustomerEmail	BookID	Title	Price	Quantity
O101	01-Oct-2026	C1	Aisha Khan	aisha@mail.com	B1, B2	Python Basics, Web Design	£20, £15	1, 2
O102	03-Oct-2026	C2	Tom Evans	tom@mail.com	B1	Python Basics	£20	1
O103	05-Oct-2026	C1	Aisha Khan	aisha@mail.com	B3	Databases	£25	1

Problem identified: order O101 contains two books, so the BookID, Title, Price and Quantity cells each hold **more than one value**. This is a **repeating group**, so the table is in UNF.

3.3 Step 2: First Normal Form (1NF)

Rules for 1NF: every cell holds one value, there are no repeating groups, and the table has a primary key.

Action: I gave each book its own row and repeated the order details on each row. The primary key is **OrderID + BookID**, because each combination appears only once (key columns are underlined).

<u>OrderID</u>	OrderDate	CustomerID	CustomerName	CustomerEmail	<u>BookID</u>	Title	Price	Quantity
O101	01-Oct-2026	C1	Aisha Khan	aisha@mail.com	B1	Python Basics	£20	1
O101	01-Oct-2026	C1	Aisha Khan	aisha@mail.com	B2	Web Design	£15	2
O102	03-Oct-2026	C2	Tom Evans	tom@mail.com	B1	Python Basics	£20	1
O103	05-Oct-2026	C1	Aisha Khan	aisha@mail.com	B3	Databases	£25	1

The table is now in 1NF. Data is still repeated, for example Aisha's details and the book "Python Basics" each appear twice.

3.4 Step 3: Second Normal Form (2NF)

Rule for 2NF: the table is in 1NF, and every non-key column depends on the **whole** primary key. This only matters when the key has more than one column, as it does here.

Finding partial dependencies. For each non-key column I asked whether I need both OrderID and BookID to find its value, or only one:

Dependency	Depends on	Type
OrderID → OrderDate, CustomerID, CustomerName, CustomerEmail	OrderID only	Partial dependency
BookID → Title, Price	BookID only	Partial dependency
(OrderID, BookID) → Quantity	Both parts	Full dependency (acceptable)

Action: I moved each group of columns into a table with the key it depends on (primary keys are underlined).

ORDER (OrderID, OrderDate, CustomerID, CustomerName, CustomerEmail)

<u>OrderID</u>	OrderDate	CustomerID	CustomerName	CustomerEmail
O101	01-Oct-2026	C1	Aisha Khan	aisha@mail.com
O102	03-Oct-2026	C2	Tom Evans	tom@mail.com
O103	05-Oct-2026	C1	Aisha Khan	aisha@mail.com

BOOK (BookID, Title, Price)

<u>BookID</u>	Title	Price
B1	Python Basics	£20
B2	Web Design	£15
B3	Databases	£25

ORDER_ITEM (OrderID, BookID, Quantity)

OrderID	BookID	Quantity
O101	B1	1
O101	B2	2
O102	B1	1
O103	B3	1

The tables are now in 2NF. "Python Basics" is stored once. Aisha's details are still stored on each of her orders.

3.5 Step 4: Third Normal Form (3NF)

Rule for 3NF: the table is in 2NF, and no non-key column depends on another non-key column. A useful summary is that every column depends on *the key, the whole key and nothing but the key*.

Finding transitive dependencies. In ORDER, the key is OrderID. However:

- OrderID → CustomerID
- CustomerID → CustomerName, CustomerEmail

Therefore CustomerName and CustomerEmail depend on OrderID only *indirectly*, through CustomerID. This is a **transitive dependency**. It is also why Aisha's details appear twice.

Action: I moved the customer columns into a new table, CUSTOMER, whose key is CustomerID (the column that decides the others). ORDER keeps CustomerID as a foreign key so each order still points to its customer.

CUSTOMER (CustomerID, CustomerName, CustomerEmail)

CustomerID	CustomerName	CustomerEmail
C1	Aisha Khan	aisha@mail.com
C2	Tom Evans	tom@mail.com

ORDER (OrderID, OrderDate, CustomerID (FK))

OrderID	OrderDate	CustomerID (FK)
O101	01-Oct-2026	C1
O102	03-Oct-2026	C2
O103	05-Oct-2026	C1

BOOK and **ORDER_ITEM** are unchanged from 2NF. Neither has a non-key column that depends on another non-key column, so all four tables are in 3NF.

4. Final Relational Schema

Table	Columns (primary key in bold, foreign key in italics)
CUSTOMER	CustomerID , CustomerName, CustomerEmail
ORDER	OrderID , OrderDate, <i>CustomerID</i>
BOOK	BookID , Title, Price
ORDER_ITEM	<i>OrderID</i> , <i>BookID</i> , Quantity (both key columns are also foreign keys)

Checking the two routes against each other. The four tables produced by normalisation (CUSTOMER, ORDER, BOOK, ORDER_ITEM) match the four entities in the ERD, with the same primary keys and foreign keys. The two independent methods agree, which gives me confidence that the design is correct.

Problems solved. Each fact is now stored once:

- Changing Aisha's email means updating one row in CUSTOMER.
- A new book can be added to BOOK before anyone orders it.
- Deleting an order removes only its rows from ORDER and ORDER_ITEM. The customer's details remain.

5. Assumptions and Evaluation

- I assumed each customer has one email address and that each order belongs to exactly one customer.
- I assumed a book appears at most once on a single order. Extra copies are recorded in Quantity.
- **Limitation:** if a book's price changes, the price on past orders would change too. A real system might store the price paid on each ORDER_ITEM row, as an extra column.
- The design was tested using three sample orders. Larger sample data would give more confidence.

6. Conclusion

The scenario was analysed to find three entities, their attributes and two relationships. The many-to-many relationship between ORDER and BOOK was resolved with the linking entity ORDER_ITEM. Normalising the order data from UNF to 3NF produced the same four tables. The final design stores every fact once and avoids the repetition, insert and delete problems of a single large table.