**Solent University**

**School:** Department of Science and Engineering

**Course:**

**Module:** Introduction to Databases (QHO442)

**Assessment:** AE2 Portfolio

**Student:**

**Date:**

**Contents page**

Part 1 – Database Design, Implementation and Integrity

Question a 3

Question b 3

Question c 3

Question d 3

Part 2 – Programming for Databases 4

Bibliography 5

# PART 1 - Database Design, Implementation and Integrity

## QUESTION:

Text

Description automatically generated

## Answer:

## *Product Review Table:*

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Product\_review\_id | INTEGER (PK, AUTO) | Unique identifier for each product review |
| product\_id | INTEGER (FK) | References Products.Product\_id |
| Shopper\_id | INTEGER (FK) | |  | | --- | |  |  |  | | --- | | References Shoppers.Shopper\_id | |
| Rating | TEXT | Star rating |
| Comment | TEXT | Shopper's written feedback |
| Review\_date\_time | DATETIME | Timestamp when the review was submitted |

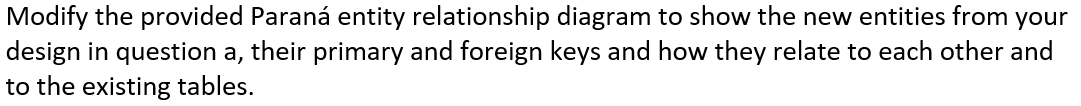
## *Seller Review Table:*

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Seller\_review\_id | INTEGER (PK, AUTO) | Unique identifier for each product review |
| seller\_id | INTEGER (FK) | References Sellers.Seller\_id |
| Shopper\_id | INTEGER (FK) | |  | | --- | |  |  |  | | --- | | References Shoppers.Shopper\_id | |
| Rating | TEXT | Star rating |
| Comment | TEXT | Shopper's written feedback |
| Review\_date\_time | DATETIME | Timestamp when the review was submitted |

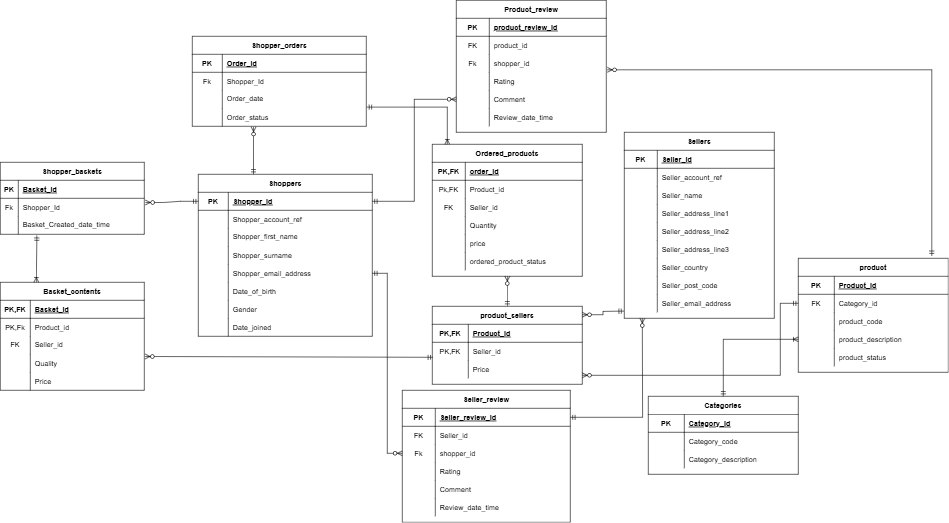
To support the functionality of storing product and seller reviews in the online shopping database, I approached the problem using the structured three-stage design process: conceptual, logical, and physical design. I determined at the conceptual design stage that the system needed to deal with two separate types of reviews, one for products and one for sellers. The star rating, a brief comment and the date and time of submission had to be stored in each review. However, these reviews had to be tied back to the shopper who wrote them, as well as the product or seller being reviewed. From this, I conceptualized two new entities: Product\_reviews and Seller\_reviews. I moved on to the logical design stage, where I defined the structure of these new tables. The fields in each table were a surrogate primary key for uniqueness (Product\_review\_id and Seller\_review\_id), foreign key fields referencing existing tables (Shopper\_id, Product\_id or Seller\_id) and fields for Rating, Comment and Review\_date\_time. In the physical design phase, I selected appropriate data types: INT for IDs, Text for star ratings (with values ranging from \* to \*\*\*\*\*), Text for comments to allow flexibility, and DATETIME for the timestamp. Normalization and well-defined foreign key relationships allow for multiple reviews per shopper, product, or seller and prevent data duplication in the database.

For the purpose of developing this design, I made several assumptions. I assumed that a shopper could possibly enter more than one review for the same product or seller after various purchases, and we could further limit that to future versions if required. I also thought that the review system was textual and star-based without the need for image upload or multimedia content. The second assumption is that reviews are visible without delay and in moderation. Thus, I didn't include status or approval fields.

## QUESTION:

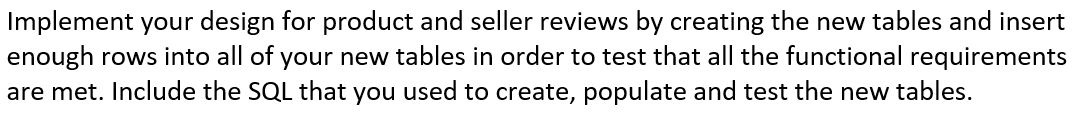


**Answer:**



**ENTITY RELATION DIAGRAM**

## QUESTION:



**Answer:**

**Table: Product\_Review**

CREATE TABLE Product\_Review (

product\_review\_id INTEGER PRIMARY KEY AUTOINCREMENT,

PRODUCT\_ID INTEGER,

SHOPPER\_ID INTEGER,

RATING TEXT,

COMMENT TEXT,

REVIEW\_DATE\_TIME DATETIME,

CONSTRAINT PRODUCT\_ID\_fk FOREIGN KEY (PRODUCT\_ID) REFERENCES products(PRODUCT\_ID),

CONSTRAINT SHOPPER\_ID\_fk FOREIGN KEY (SHOPPER\_ID) REFERENCES shoppers(SHOPPER\_ID)

);

The customer feedback on products is stored in the Product\_Review table, which associates the reviews with shoppers' products through foreign keys. It consists of a unique product\_review\_id, PRODUCT\_ID, SHOPPER\_ID, RATING, COMMENT, and REVIEW\_DATE\_TIME. Foreign key constraints guarantee that data in the products and shoppers tables is consistent. Then, INSERT statements show reviews for different products, including ratings and comments.

**Insert into Product\_Review**

INSERT INTO Product\_Review (PRODUCT\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (3007676, 10000, '5 Stars', 'Amazing camera! Very happy with the purchase.', '2025-05-06 10:30:00');

INSERT INTO Product\_Review (PRODUCT\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (3007779, 10005, '4 Stars', 'Good tablet, but battery life could be better.', '2025-05-06 11:00:00');

INSERT INTO Product\_Review (PRODUCT\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (3007905, 10011, '3 Stars', 'Decent console, but I prefer PlayStation.', '2025-05-06 12:00:00');

INSERT INTO Product\_Review (PRODUCT\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

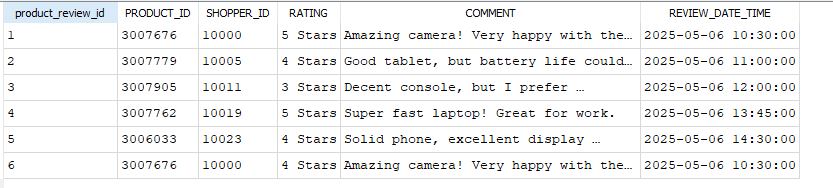
VALUES (3007762, 10019, '5 Stars', 'Super fast laptop! Great for work.', '2025-05-06 13:45:00');

INSERT INTO Product\_Review (PRODUCT\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (3006033, 10023, '4 Stars', 'Solid phone, excellent display quality.', '2025-05-06 14:30:00');

INSERT INTO Product\_Review (PRODUCT\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (3007676, 10000, '4 Stars', 'Amazing camera! Very happy with the purchase.', '2025-05-06 10:30:00');



**Product\_Review\_Table**

**Table: SELLER\_Review**

CREATE TABLE SELLER\_Review (

SELLER\_review\_id INTEGER PRIMARY KEY AUTOINCREMENT,

SELLER\_ID INTEGER,

SHOPPER\_ID INTEGER,

RATING TEXT,

COMMENT TEXT,

REVIEW\_DATE\_TIME DATETIME,

CONSTRAINT SELLER\_ID\_fk FOREIGN KEY (SELLER\_ID) REFERENCES SELLERS(SELLER\_ID),

CONSTRAINT SHOPPER\_ID\_fk FOREIGN KEY (SHOPPER\_ID) REFERENCES shoppers(SHOPPER\_ID)

);

The SELLER\_Review table stores shopper reviews for a seller. Each review has a unique ID, seller ID, shopper ID, rating, comment and review timestamp. Referential integrity is enforced in the table by foreign key constraints linking seller and shopper IDs to other tables. The unique\_shopper\_review constraint also helps improve the quality of data by ensuring that a shopper cannot submit multiple reviews for the same seller. This structure keeps ratings clean and meaningful so sellers can easily track them. It makes the database more reliable, and customer reviews more trustworthy, allowing for better decision-making.

**Insert into SELLER\_Review**

INSERT INTO SELLER\_Review (SELLER\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (200000, 10000, '5 Stars', 'Fast delivery and great customer service.', '2025-05-06 10:40:00');

INSERT INTO SELLER\_Review (SELLER\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (200005, 10005, '4 Stars', 'Item arrived on time but packaging could be improved.', '2025-05-06 11:30:00');

INSERT INTO SELLER\_Review (SELLER\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

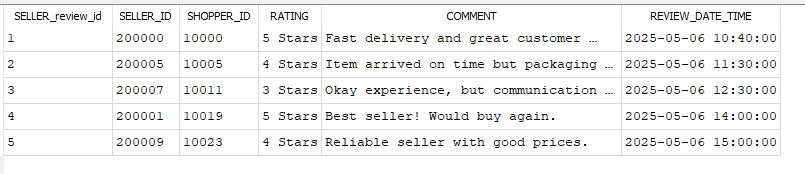
VALUES (200007, 10011, '3 Stars', 'Okay experience, but communication was slow.', '2025-05-06 12:30:00');

INSERT INTO SELLER\_Review (SELLER\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (200001, 10019, '5 Stars', 'Best seller! Would buy again.', '2025-05-06 14:00:00');

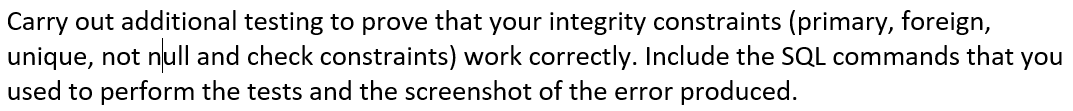
INSERT INTO SELLER\_Review (SELLER\_ID, SHOPPER\_ID, RATING, COMMENT, REVIEW\_DATE\_TIME)

VALUES (200009, 10023, '4 Stars', 'Reliable seller with good prices.', '2025-05-06 15:00:00');



**SELLER\_Review\_Table**

1. **QUESTION:**



**Answer:**

**Test 1:**

SELECT pr.product\_review\_id, p.PRODUCT\_ID, p.product\_code , s.SHOPPER\_ID, s.shopper\_first\_name , s.shopper\_surname,

pr.RATING, pr.COMMENT, pr.REVIEW\_DATE\_TIME

FROM Product\_Review pr

JOIN products p ON pr.PRODUCT\_ID = p.PRODUCT\_ID

JOIN shoppers ON pr.SHOPPER\_ID = s.SHOPPER\_ID;

This SQL query retrieves detailed product review information by joining three tables: Product\_Review, products, and shoppers. It selects the review IDs, product details, shopper details, ratings, comments and review timestamp. The JOIN clauses connect the reviews to the products and shoppers so that we can see the whole picture of customer feedback. This query effectively integrates data from various tables. It makes it more efficient to report on shopper opinions and product performance without having to put the data in a structured relationship within the database.



**Test 2:**

SELECT sr.SELLER\_review\_id, se.SELLER\_ID, se.SELLER\_NAME, s.SHOPPER\_ID, s.shopper\_first\_name , s.shopper\_surname ,

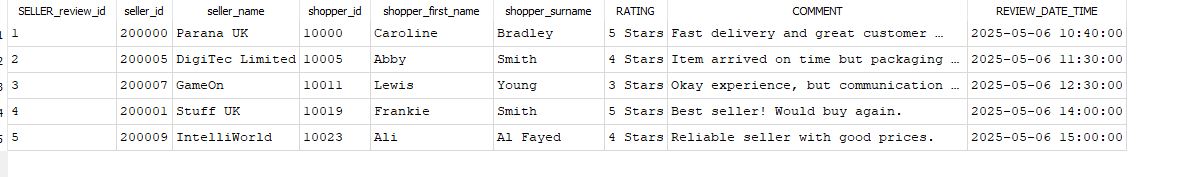
sr.RATING, sr.COMMENT, sr.REVIEW\_DATE\_TIME

FROM SELLER\_Review sr

JOIN sellers se ON sr.SELLER\_ID = se.SELLER\_ID

JOIN shoppers ON Sr.SHOPPER\_ID = s.SHOPPER\_ID;

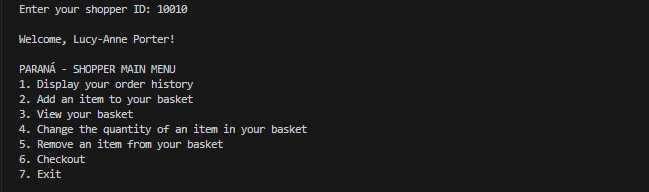
This SQL query retrieves detailed seller review information by joining three tables: SELLER\_Review, sellers, and shoppers. It chooses review IDs, seller details, shopper details, ratings, comments, and review timestamps. The JOIN clauses make sure that every review is associated with its respective seller and shopper, making the feedback complete. This query integrates data from multiple tables and helps businesses analyze shopper opinions about sellers and ratings and maintain structured relationships in the database. This enables better seller performance evaluation and more accurate customer feedback.



# PART 2 – Programming for Databases

## Shopper Identification and Main Menu Implementation

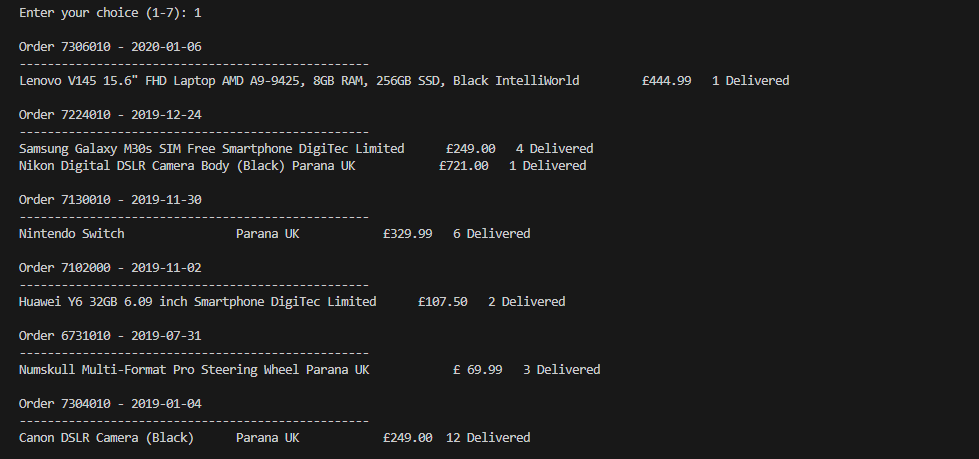
The shopper identification system is successfully implemented in the program. It immediately prompts users to enter their shopper ID, which is validated against the database upon execution. To validate the process, the shopper's table is queried to ensure the ID exists and the shopper’s full name is retrieved to be used in the welcome message. In case an invalid ID is entered, the program prints an appropriate error message and terminates so that unauthorized access is prevented. In addition, the main menu is clear and easy to use and exactly follows the specifications, with all seven options correctly labelled. All functionality remains easily available, with the menu system running in a continuous loop until the user explicitly exits.

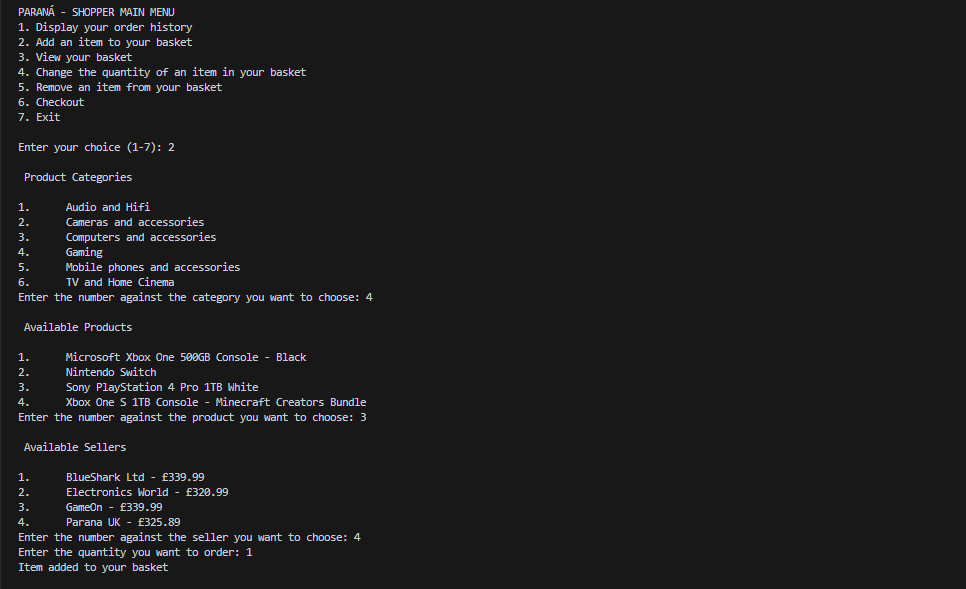


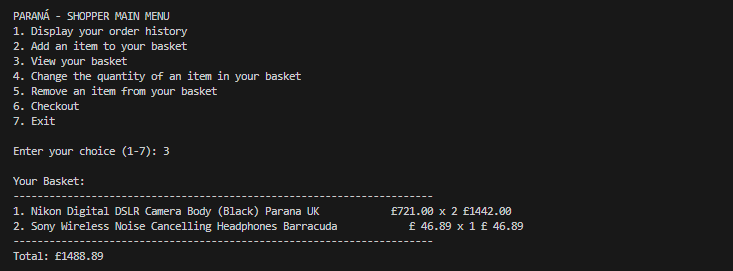


## Order History and Basket Management Features

The functionality of displaying order history comprehensively meets all the requirements. It returns the selected complete order information (such as product details, seller information, pricing, and status) by executing an SQL query that joins many tables. The data elements are aligned, and orders are clearly separated. If there is no order, it displays a message rather than an empty result. The add-item process for basket management is a logical sequence of category selection, product selection, seller selection, and quantity specification, with strong input validation at each step. The view basket feature lists items numerically with all details required and calculates the running total exactly as specified in the requirements document.

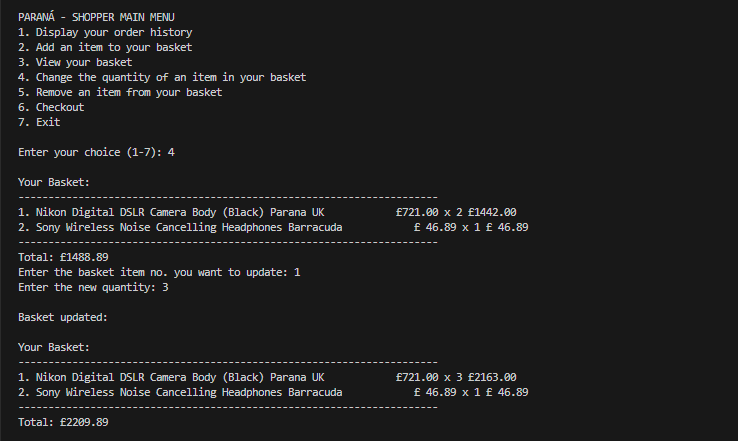


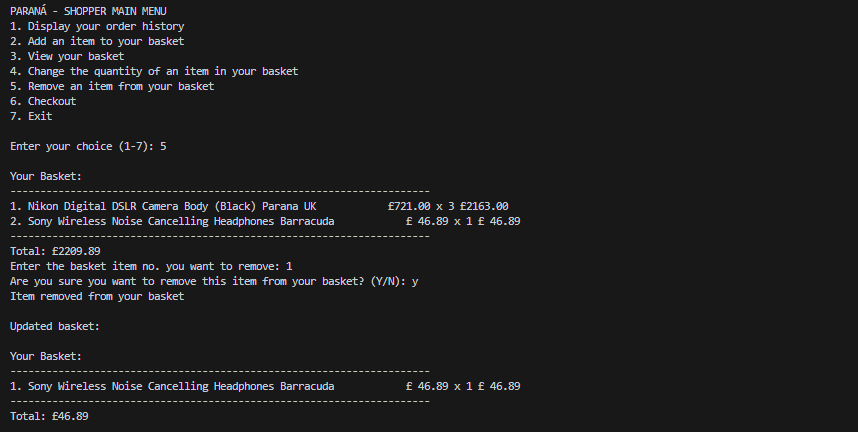




## Basket Modification Functionality Implementation

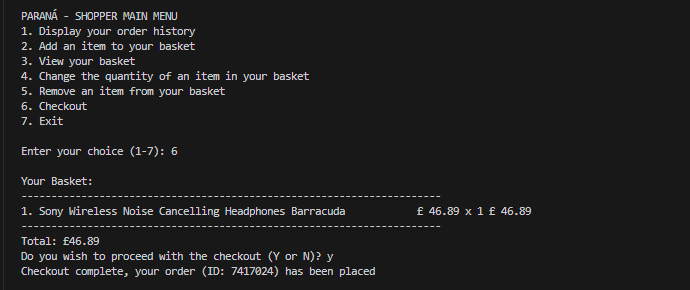
Quantity modification gives users intuitive control over how much of an item they want to add to their basket. It is invoked to check for empty baskets, and if none exists, it returns to the menu to prevent unnecessary operations. In the case of baskets with contents, it shows all items with reference numbers and handles both single-item and multi-item cases appropriately. With zero or negative values, the system enforces strict quantity validation and rejects them with clear error messages. It automatically updates the updated basket with the recalculated totals, showing users right away what their modified basket looks like. The pattern for the item removal functionality is similar, except with the addition of another confirmation step to prevent them from being removed accidentally. Both features employ appropriate transaction handling to maintain data integrity, and their user flows are exactly those specified in the requirements specification.





## Checkout Process and Transaction Management

The implementation of the checkout is a robust solution for finalizing orders while keeping data consistent. First, it does standard basket content and user confirmation checks, and then it proceeds with the transaction. The system uses proper database transaction methods that will group all related operations into a single atomic unit that either succeeds entirely or fails entirely in the case of errors. This also prevents partial updates that might leave the database in an inconsistent state. All basket items are properly transferred to ordered products, order records are created with appropriate status markers, and the basket is cleared.



# Appendix:

