BANL-6430-02 COURSE PROJECT

Fall 2024

Name: Anu Janagam (00887091)

ajana3@unh.newhaven.edu

1.One to two paragraphs explaining your database solution

My database solution was created for an online shopfront that is Amazon.com. It effectively arranges customer, product, order, address, and review data. In order to enable website functions like product display, order processing, and client feedback collection, this relational database makes it simple to maintain and query data. The database is normalised with distinct associations between elements like customers, orders, and items to reduce redundancy and enhance efficiency.

2. ER diagram design and explanation

A diagram of a process

Description automatically generated

Explanation of Your ERD:

Entities:

Customer

Order

Products

Address

Reviews

Each entity has its own attributes:

Customer: CustomerID (PK), UserName, Email, Password, Contact number

Order: OrderID (PK), Address, Total Price, Order Status

Products: ProductID (PK), Product name, Price, Description, Stock quantity

Address: AddressID (PK), Country, City, State

Reviews: ReviewID (PK), Rating, Customer ID (FK), Product ID (FK)

This structure aligns with the typical requirements of an e-commerce system.

Relationships:

Customer places Order (1:N relationship)

A customer can place multiple orders.

Order contains Products (N:M relationship)

An order can have multiple products, and a product can appear in multiple orders.

Products delivered to Address (logical association)

The "has" relationship correctly shows an address linked to an order.

Customer writes Reviews (1:N relationship)

A customer can write reviews for products.

Reviews are associated with Products (1:N relationship)

A review is tied to a specific product.

3.Relational schema diagram showing tables and relationships

A diagram of a product

Description automatically generated

4.Screenshots of the tables, including the data that has been populated in them  
 o Each table should have at least 6 sample records.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

5. Using two or three tables inner joined, write a query that includes aggregation (e.g.  
AVG, SUM, etc.) with GROUP BY. Include SQL statement for this query in your report.  
Provide a description what your query does.

**SELECT**

*p*.***Product\_name***,

**SUM**(*od*.***Quantity***) **AS** *Total\_Quantity\_Sold*,

**AVG**(*p*.***Price***) **AS** *Average\_Price*

**FROM**

Products *p*

**INNER** **JOIN**

OrderDetails *od* **ON** *p*.***ProductID*** = *od*.***ProductID***

**INNER** **JOIN**

Orders *o* **ON** *od*.***OrderID*** = *o*.***OrderID***

**GROUP** **BY**

*p*.***Product\_name***;

**Description:**

INNER JOIN:

uses the ProductID to join the OrderDetails table (od) with the Products table (p). This gives us access to the order and product details.

To obtain the full order information, it additionally combines the OrderDetails table (od) with the Orders table (o) according to the OrderID.

Combination:

SUM(od.Quantity): This determines how much of each product was sold overall across all orders.

AVG(p.Price): This determines each product's average cost. It is included to demonstrate how the aggregate operates, even if the price may not change in the Products table.

GROUP BY:

Since the data is arranged according to Product\_name, the outcomes will be combined for every unique product.

6. Now answer the same question in Question 5 but this time use R code. Include R file with your submission.

A screenshot of a computer

Description automatically generated

7. Create one (1) VIEW in your DBstudent \_ \_ \_ database.  
Some options for a view can be found below:  
- top 3 products sold,  
- products sold in most transactions,  
- days with most sales, etc.  
Paste the code for the VIEW. Provide a description for your VIEW

**CREATE** **VIEW** Top3ProductsSold **AS**

**SELECT**

p.ProductID,

p.Product\_name,

**SUM**(od.Quantity) **AS** TotalQuantitySold

**FROM**

Products p

**INNER** **JOIN**

OrderDetails od **ON** p.ProductID = od.ProductID

**GROUP** **BY**

p.ProductID, p.Product\_name

**ORDER** **BY**

TotalQuantitySold **DESC**

**LIMIT** 3;

**SELECT** \* **FROM** Top3ProductsSold *tps* ;

**Description of the View**

a) View Definition (CREATE VIEW):  
The query creates a view named Top3ProductsSold. A view stores the result of a SQL query as a virtual table that can be referenced like a regular table. It does not store actual data but dynamically generates results when accessed.

b) Tables Involved:

* **Products (p)**: Contains product details such as ProductID and Product\_name.
* **OrderDetails (od)**: Records the quantity of each product sold in individual orders.

c) Join Operation (INNER JOIN):  
The query connects the Products and OrderDetails tables using their common column, ProductID.  
This ensures that the quantity of products sold (OrderDetails) is linked to their corresponding product names (Products).

d) Aggregate Function (SUM(od.Quantity)):

* SUM(od.Quantity) calculates the total quantity sold for each product across all orders.
* This total represents the product's cumulative sales.

e)Grouping (GROUP BY):

* GROUP BY p.ProductID, p.Product\_name groups the data by ProductID and Product\_name.
* Each unique product forms a group, and the SUM function calculates the total quantity sold within each group.

f) Sorting (ORDER BY):

* ORDER BY TotalQuantitySold DESC sorts the results in descending order of the total quantity sold.
* The product with the highest sales appears first.

g) Limiting Results (LIMIT 3):

* LIMIT 3 restricts the output to only the top 3 products based on their total sales volume.

3. View Functionality:

Once created, the Top3ProductsSold view provides:

* **Dynamic Updates**: Whenever the underlying tables (Products and OrderDetails) are modified, the view automatically reflects the changes.
* **Ease of Access**: Users can simply query the view (SELECT \* FROM Top3ProductsSold) instead of rewriting the aggregation log

By creating this view, businesses gain an efficient and scalable way to monitor their top-selling products, enabling data-driven decision-making.

A screenshot of a computer

Description automatically generated