Online Retail Management System

A COURSE PROJECT REPORT

By

Sanjay Aaditya S (RA2111003011180)

Under the guidance of **Dr. Sandhia G K**

In partial fulfillment for the Course

18CSC303J-Database Management Systems

in

School of Computing

FACULTY OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chengalpattu District

APRIL 2024.

Acknowledgement

We would like to express our gratitude to our Professor, **Dr. Sandhia G K** who gave me the golden opportunity to do this wonderful project on the topic **"ONLINE RETAIL MANAGEMENT SYSTEM"** which also helped me in doing a lot of research and I came to know about so many new things I am really thankful to her.

I am also thankful to all the other faculty, teaching and non-teaching staff members of our department for their kind co-operation and help.

Lastly, I would also like to thank my friends who helped me a lot in finishing this project within the limited time. I am making this project not only for marks but to also increase my knowledge.

INDEX

Introduction

The Online Retail Shop Database Management System (DBMS) serves as the backbone of an online retail store's operations. It is designed to efficiently manage various aspects of the retail business, including inventory management, order processing, customer interaction, and sales tracking. By providing a centralized platform for managing store data, the DBMS enhances the overall efficiency and effectiveness of the retail operations.

System Requirements

1. Operating System: The project can run on any operating system that supports the Oracle Database system, such as Windows, macOS, or Linux.
2. Database Management System: You need to have Oracle Database installed on your system to create and manage the database tables and execute SQL queries and procedures.
3. Database Connectivity: Ensure that your system has proper connectivity to the Oracle Database. You'll need appropriate drivers or libraries to connect your application to the database.
4. Oracle Database: Make sure you have Oracle Database installed and configured on your system. You may use Oracle Database Express Edition (XE) for development purposes, which is a free version of Oracle Database.
5. Oracle SQL Developer (Optional): Oracle SQL Developer is a graphical tool for database development. You can use it to interact with the Oracle Database, write SQL queries, and execute procedures. While not strictly required, it can be helpful for managing the database.
6. Oracle Client (Optional): If you're connecting to a remote Oracle Database, you may need to install Oracle Client software on your system to establish the connection.
7. Development Environment: You can use any text editor or integrated development environment (IDE) for writing SQL queries, procedures, and triggers. IDEs like Oracle SQL Developer, PL/SQL Developer, or Visual Studio Code with appropriate extensions can be used.
8. Hardware Requirements: The hardware requirements depend on the size and complexity of your database and the expected workload. Ensure that your system has sufficient RAM, CPU, and disk space to handle the Oracle Database and the associated tools.
9. Oracle Database Administrator (DBA) Skills (Optional): If you're responsible for setting up and managing the Oracle Database, having knowledge and skills in database administration is beneficial. This includes tasks like user management, backup and recovery, performance tuning, etc.

Database Design

The database is structured to accommodate the specific needs of an online retail shop. It consists of two main tables: retail and orders.

Retail Table: This table stores detailed information about the products available for sale in the retail store. It includes attributes such as item name, quantity available, unit price, and current availability status.

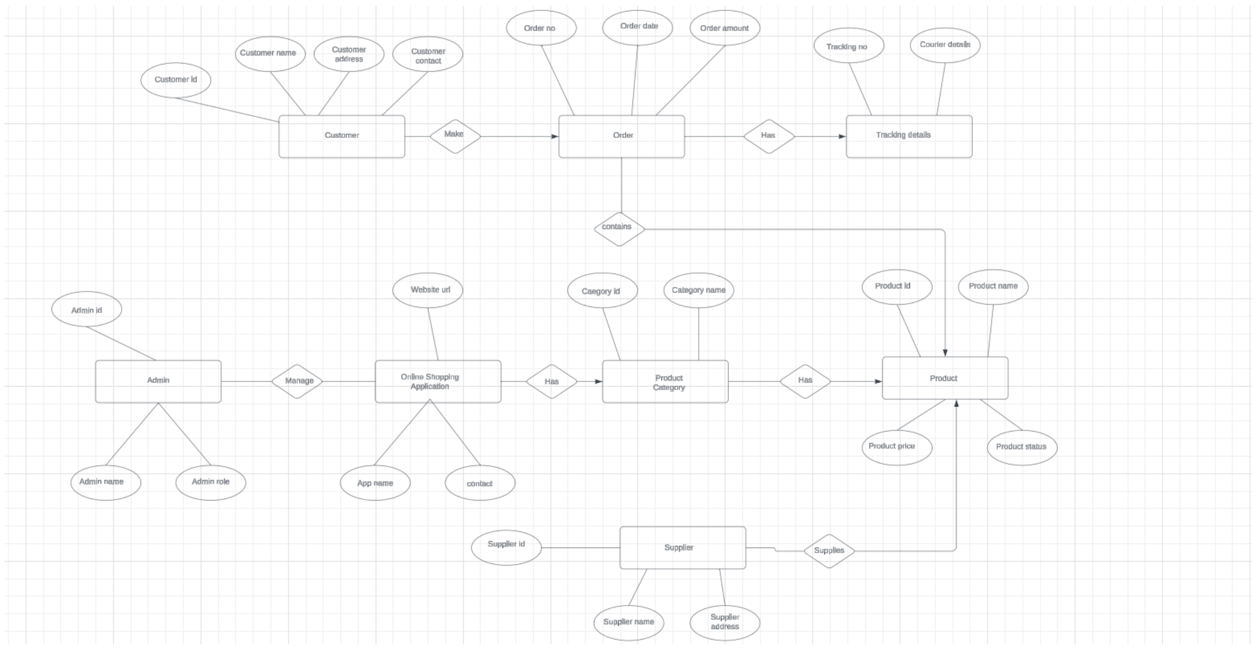
Orders Table: The orders table records essential details about customer orders. It captures information such as customer name, ordered item, chosen payment method, and order placement date.

Table Structure:

1. Retail Table:
    - S_NO (Serial Number): This column serves as the primary key for the retail table. It uniquely identifies each record in the table. It is of type INT (integer).
    - ITEM_NAME: This column stores the name of the item sold in the retail store. It is of type VARCHAR(100), which means it can store variable-length character strings up to 100 characters long.
    - QTY (Quantity): This column stores the quantity of each item available in the retail inventory. It is of type INT (integer).
    - PRICE: This column stores the price of each item. It is of type DECIMAL(10, 2), which means it can store decimal numbers with a precision of 10 digits, with 2 digits after the decimal point.
    - IN_STOCK: This column indicates whether the item is currently in stock or not. It is of type INT (integer), where 1 typically represents "in stock" and 0 represents "out of stock".

2. Orders Table:
    - S_NO (Serial Number): This column serves as the primary key for the orders table. It uniquely identifies each order placed by a customer. It is of type INT (integer).
    - CUSTOMER_NAME: This column stores the name of the customer who placed the order. It is of type VARCHAR(100).
    - ORDERED_ITEM: This column stores the name of the item that the customer ordered. It is of type VARCHAR(100).
    - PAYMENT_METHOD: This column stores the payment method used by the customer for the order. It is of type VARCHAR(50).
    - ORDERED_DATE: This column stores the date on which the order was placed. It is of type DATE.

Entity Relationship Diagram:



Procedures and Functions:

The implemented procedures and functions facilitate seamless interaction with the database and streamline essential retail operations:

add_item_to_retail: This procedure allows store administrators to add new items to the retail inventory by specifying the item name, price, available quantity, and availability status.

process_order: Customers can place orders using this procedure, which inserts order details into the orders table, including customer name, ordered item, payment method, and order date.

update_availability: Store administrators can update the quantity of items in the retail inventory using this procedure, ensuring accurate tracking of available stock levels.

get_total_order_price: This function calculates and returns the total price of all orders placed by a specific customer, aiding in sales analysis and customer relationship management.

check_item_availability: Customers and store administrators can use this function to check the availability status of a particular item in the retail inventory, helping to manage customer expectations and inventory levels.

retrieve_order_details: This procedure retrieves and displays comprehensive order details, including customer information, ordered items, and retail inventory status, providing insights into order fulfillment and inventory management processes.

Triggers:

Triggers are employed to automate specific actions within the database environment:

update_retail_quantity: This trigger automatically updates the quantity of an item in the retail inventory after an order is placed, ensuring real-time synchronization between order processing and inventory management.

log_order_details: After an order is successfully placed, this trigger logs relevant order details into a separate table, facilitating order tracking, analysis, and audit trails.

1. **update_retail_quantity Trigger:**

   This trigger updates the quantity of an item in the retail inventory after an order is placed.

   Usage Scenario: A customer places an order for a "Laptop," reducing the available quantity in the retail inventory.

   Expected Outcome: The trigger automatically updates the quantity of the "Laptop" in the retail Inventory after the order is placed.

2. **log_order_details Trigger**:

   This trigger logs order details into a separate table after an order is placed.

   Usage Scenario: A customer named "John" places an order for a "Gaming Console."

   Expected Outcome: After the order is placed, relevant order details such as customer name, order date, and price are logged into a separate table named order_logs.

Usage:

Users interact with the Online Retail Shop DBMS through its user-friendly interface or programmatically using SQL queries. They can leverage the provided procedures and functions to perform various tasks, such as adding new products, processing orders, updating inventory, retrieving order information, and checking item availability. The system's intuitive design and robust functionality empower users to efficiently manage retail operations and deliver exceptional customer experiences.

Adding New Items to Retail Inventory:

```
add_item_to_retail('Smartwatch', 199.99, 1, 10);
```

```
Item added successfully.
```

Placing Customer Orders:

```
process_order('Alice', 'Laptop', 'Credit Card', SYSDATE);
```

```
Order processed successfully.
```

Checking Item Availability:

```
SELECT check_item_availability('Smartphone') AS AVAILABILITY FROM DUAL;
```

```
AVAILABILITY
------------
Available
```

Retrieving Order Details:

```
retrieve_order_details;
```

```
Order Details - Customer: Alice, Date: 28-APR-2024, Price: 100000
Retail Details - Item: Laptop, Price: 100000, Availability: 1, Quantity: 21
```

Updating Inventory Quantity:

```
update_availability('Watches', 70);
```

```
Quantity updated successfully.
```

Conclusion:

In conclusion, the project aimed to develop a comprehensive database system for managing an online retail shop's inventory and order processing. The system was designed to provide functionalities for adding new items to the inventory, processing customer orders, updating item availability, and generating reports on order details and product availability.

The project successfully achieved its objectives by implementing the following key components:

1. Database Schema Design: The project began with designing an appropriate database schema to represent the retail shop's inventory and order management system. This included creating tables such as retail to store product information and orders to store customer orders.
2. Data Population: Sample data was inserted into the database tables to simulate a realistic scenario. This data included information about various retail products, such as laptops, smartphones, gaming consoles, and more, along with customer orders placed for these products.
3. Stored Procedures and Functions: Several stored procedures and functions were created to encapsulate complex business logic and database operations. These procedures included functionalities for adding items to the inventory, processing customer orders, updating item availability, calculating total order prices, and checking item availability.
4. Triggers: Triggers were implemented to automate certain actions in response to database events. For example, an AFTER INSERT trigger was created to update the quantity of items in the inventory table after a new order is placed.
5. Error Handling: The system incorporated error handling mechanisms to gracefully handle exceptions and provide informative error messages to users in case of any issues during database operations.

Future Enhancements:

While the current implementation of the project fulfills the basic requirements of a retail inventory and order management system, there are several areas where future enhancements and refinements can be made:

User Authentication and Authorization: Implementing user authentication and authorization mechanisms to control access to sensitive data and functionalities based on user roles and permissions.

Integration with E-commerce Platforms: Integrating the system with popular e-commerce platforms to enable online sales and reach a broader customer base.

Advanced Reporting and Analytics: Enhancing reporting capabilities to generate detailed analytics and insights into sales trends, customer behavior, and inventory performance.

Inventory Optimization: Implementing advanced algorithms for inventory optimization, including demand forecasting, stock replenishment strategies, and inventory turnover analysis.

Mobile Compatibility: Developing a mobile-friendly version of the user interface to enable access from smartphones and tablets, catering to the growing trend of mobile shopping.

Code:

```
CREATE TABLE retail (
    S_NO INT PRIMARY KEY,
    ITEM_NAME VARCHAR(100),
    QTY INT,
    PRICE DECIMAL(10, 2),
    IN_STOCK INT
);

INSERT INTO retail (S_NO, ITEM_NAME, QTY, PRICE, IN_STOCK) VALUES
(1, 'Laptop', 21, 100000, 1),
(2, 'Smart Phone', 21, 40000, 1),
(3, 'Gaming Console', 22, 50000, 1),
(4, 'Portable Speaker', 10, 4999, 1),
(5, 'Microwave', 15, 9999, 1),
(6, 'Watches', 20, 3299, 1),
(7, 'Smart Watches', 8, 8299, 1),
(8, 'Headphones', 11, 7899, 1);

CREATE TABLE orders (
    S_NO INT PRIMARY KEY,
    CUSTOMER_NAME VARCHAR(100),
    ORDERED_ITEM VARCHAR(100),
    PAYMENT_METHOD VARCHAR(50),
    ORDERED_DATE DATE
);
INSERT INTO orders (S_NO, CUSTOMER_NAME, ORDERED_ITEM, PAYMENT_METHOD, ORDERED_DATE) VALUES
(1, 'Thusr', 'Laptop', 'Credit Card', '2024-04-28'),
(2, 'Sanjay', 'Smart Watches', 'PayPal', '2024-04-28'),
(3, 'Joesam', 'Gaming Console', 'Debit Card', '2024-04-29');


SELECT o.CUSTOMER_NAME, o.ORDERED_ITEM, r.PRICE
FROM orders o
INNER JOIN retail r ON o.ORDERED_ITEM = r.ITEM_NAME;
SELECT o.CUSTOMER_NAME, o.ORDERED_ITEM, COALESCE(r.PRICE, 'Not Available') AS PRICE
FROM orders o
```

```
LEFT JOIN retail r ON o.ORDERED_ITEM = r.ITEM_NAME;

CREATE OR REPLACE PROCEDURE add_item_to_retail (
    p_item_name IN VARCHAR2,
    p_price IN NUMBER,
    p_in_stock IN NUMBER,
    p_quantity IN NUMBER
)
IS
BEGIN
    INSERT INTO retail (ITEM_NAME, PRICE, IN_STOCK, QTY)
    VALUES (p_item_name, p_price, p_in_stock, p_quantity);
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Item added successfully.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END add_item_to_retail;

CREATE OR REPLACE PROCEDURE process_order (
    p_customer_name IN VARCHAR2,
    p_ordered_item IN VARCHAR2,
    p_payment_method IN VARCHAR2,
    p_order_date IN DATE
)
IS
BEGIN
    INSERT INTO orders (CUSTOMER_NAME, ORDERED_ITEM, PAYMENT_METHOD, ORDERED_DATE)
    VALUES (p_customer_name, p_ordered_item, p_payment_method, p_order_date);
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Order processed successfully.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END process_order;

CREATE OR REPLACE PROCEDURE update_availability (
    p_item_name IN VARCHAR2,
  p_new_quantity IN NUMBER
)
IS
BEGIN
  UPDATE retail
  SET QTY = p_new_quantity
  WHERE ITEM_NAME = p_item_name;
  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Quantity updated successfully.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END update_availability;

CREATE OR REPLACE FUNCTION get_total_order_price (
  p_customer_name IN VARCHAR2
)
RETURN NUMBER
IS
  v_total_price NUMBER;
BEGIN
  SELECT NVL(SUM(price), 0)
  INTO v_total_price
  FROM orders
  WHERE CUSTOMER_NAME = p_customer_name;

  RETURN v_total_price;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 0; -- If no orders found for the customer, return 0
  WHEN OTHERS THEN
    RETURN NULL; -- Handle other exceptions as needed
END get_total_order_price;

CREATE OR REPLACE FUNCTION check_item_availability (
  p_item_name IN VARCHAR2
)
RETURN VARCHAR2
```

```
IS
   v_availability VARCHAR2(20);
BEGIN
   SELECT CASE
         WHEN IN_STOCK > 0 THEN 'Available'
         ELSE 'Out of Stock'
        END
   INTO v_availability
   FROM retail
   WHERE ITEM_NAME = p_item_name;

   RETURN v_availability;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN 'Not Found'; -- If item not found, return 'Not Found'
   WHEN OTHERS THEN
      RETURN NULL; -- Handle other exceptions as needed
END check_item_availability;
CREATE OR REPLACE PROCEDURE retrieve_order_details IS
   CURSOR order_cursor IS
      SELECT CUSTOMER_NAME, ORDERED_DATE, PRICE
      FROM orders;

   CURSOR retail_cursor IS
      SELECT ITEM_NAME, PRICE, IN_STOCK, QTY
      FROM retail;

   v_customer_name orders.CUSTOMER_NAME%TYPE;
   v_order_date orders.ORDERED_DATE%TYPE;
   v_price orders.PRICE%TYPE;

   v_item retail.ITEM_NAME%TYPE;
   v_price retail.PRICE%TYPE;
   v_availability retail.IN_STOCK%TYPE;
   v_quantity retail.QTY%TYPE;
BEGIN
   -- Open and fetch from orders cursor
   OPEN order_cursor;
```

```
    LOOP
        FETCH order_cursor INTO v_customer_name, v_order_date, v_price;
        EXIT WHEN order_cursor%NOTFOUND;

        -- Process order data
        DBMS_OUTPUT.PUT_LINE('Order Details - Customer: ' || v_customer_name || ', Date: ' || TO_CHAR(v_order_date, 'DD-MON-YYYY') ||
', Price: ' || v_price);
    END LOOP;
    CLOSE order_cursor;

    -- Open and fetch from retail cursor
    OPEN retail_cursor;
    LOOP
        FETCH retail_cursor INTO v_item, v_price, v_availability, v_quantity;
        EXIT WHEN retail_cursor%NOTFOUND;

        -- Process retail data
        DBMS_OUTPUT.PUT_LINE('Retail Details - Item: ' || v_item || ', Price: ' || v_price || ', Availability: ' || v_availability || ', Quantity: ' ||
v_quantity);
    END LOOP;
    CLOSE retail_cursor;
END retrieve_order_details;

CREATE OR REPLACE PROCEDURE process_order (
    p_customer_name IN VARCHAR2,
    p_ordered_item IN VARCHAR2,
    p_payment_method IN VARCHAR2,
    p_order_date IN DATE
)
IS
BEGIN
    -- Attempt to insert the order into the orders table
    INSERT INTO orders (CUSTOMER_NAME, ORDERED_ITEM, PAYMENT_METHOD, ORDERED_DATE)
    VALUES (p_customer_name, p_ordered_item, p_payment_method, p_order_date);

    -- Commit the transaction
    COMMIT;

    -- Output success message
    DBMS_OUTPUT.PUT_LINE('Order processed successfully.');
EXCEPTION
    -- Handle constraint violation exceptions
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Error: Duplicate order detected. Please try again.');
    -- Handle other exceptions
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END process_order;

CREATE OR REPLACE TRIGGER update_retail_quantity
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    UPDATE retail
    SET QTY = QTY - 1
    WHERE ITEM_NAME = :NEW.ORDERED_ITEM;
END;
/

CREATE OR REPLACE TRIGGER log_order_details
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    INSERT INTO order_logs (CUSTOMER_NAME, ORDER_DATE, PRICE)
    VALUES (:NEW.CUSTOMER_NAME, :NEW.ORDERED_DATE, :NEW.PRICE);
END;
/
```

References

https://www.w3schools.com/sql/

https://en.wikipedia.org/wiki/SQL

https://www.geeksforgeeks.org/sql-tutorial/

https://www.tutorialspoint.com/sql/index.htm

https://www.javatpoint.com/sql-tutorial

https://www.oracle.com/in/database/

https://www.oracle.com/in/database/technologies/appdev/sql.html

https://www.sqltutorial.org/

https://github.com/Krushna153/Online-Shopping-Management-System

https://www.lucidchart.com/pages/examples/er-diagram-tool

https://chat.openai.com/