GROUP: A

PROJECT: PIZZATAKEOUT

Oracle Pizza Takeaway Database System

Introduction

This paper provides a plan and development of a database to be used by a pizza takeaway shop business. Customers, orders, type of pizzas, materials used, and suppliers information are found in the database. In other instance, it means that ensures that information flows well in an organization without compromising on data accuracy and capacity. The report provides the Entity Relationship Diagram (ERD), discussion about the aspects of normalization, example SQL queries, and discussion about how to prevent SQL injection attacks. Furthermore, the system is expected to process and manage the huge amount of data and at the same time should should provide efficiency and dependability.

Entity Relationship Diagram abbreviated as ERD.

The Entity Relationship Diagram (ERD) of Pizza Takeaway Shop Database is generated using Crow’s foot notation. It reflects the fact that Customers place orders, Pizzas contain ingredients affecting Customers and Order, Ingredients include Suppliers and Pizzas, etc. The relationships guarantee the system to track interactions with customers, orders and inventory management appropriately. Below are the key relationships:

- \*Customers to Orders\*: In one order, a customer can purchase several products (1 to Many).

- \*Orders to Pizzas\*: It can have multiple pizzas per order through use of Order\_Items table which is also of Many to Many relationship.

- \*Pizzas to Ingredients\*: This is Many to Many and each pizza can contain let’s say, 2 or more ingredients via Pizza\_Ingredients.

- \*Suppliers to Ingredients\*: Each supplier can offer many ingredients (One to Many).

An example of such relationships is included in the detailed ERD diagram available in the repository.

Normalization Considerations

These steps are very useful in normalizing a database so as to come up with efficient and an efficient database system. For the Pizza Takeaway Database, normalization was applied in three steps:

1. First Normal Form (1NF):

Tables were intended in a way that all the fields hold atomic data. For example, fields like Name, Phone and Address were split into different fields as in the table below.

2. Second Normal Form (2NF):

- Partial dependencies were removed by making certain that all non-key field relies on all the fields in the primary key. For instance, you would see that in relation to pizza attributes such as name, size, and price, PizzaID is the unique determiner.

3. Third Normal Form (3NF):  
- Transitive dependencies were removed. For instance, pizza prices depend directly on PizzaID and not on other non-key attributes. This ensures data integrity and reduces redundancy.

## SQL Query for Placing an Order

The following SQL queries demonstrate how an order is placed and associated with pizzas. This example covers inserting a new order, adding items to the order, and retrieving the order summary for reporting purposes.

1. \*Insert a new order:\*  
```sql  
INSERT INTO Orders (CustomerID, OrderDate, TotalAmount)   
VALUES (1, NOW(), 25.00);  
  
SET @last\_order\_id = LAST\_INSERT\_ID();  
```  
2. \*Add pizzas to the order:\*  
```sql  
INSERT INTO Order\_Items (OrderID, PizzaID, Quantity, Subtotal)   
VALUES   
(@last\_order\_id, 2, 1, 12.50),  
(@last\_order\_id, 3, 1, 12.50);  
```  
3. \*Retrieve order summary:\*  
```sql  
SELECT o.OrderID, c.Name AS CustomerName, p.Name AS PizzaName, oi.Quantity, oi.Subtotal   
FROM Orders o   
JOIN Customers c ON o.CustomerID = c.CustomerID   
JOIN Order\_Items oi ON o.OrderID = oi.OrderID   
JOIN Pizzas p ON oi.PizzaID = p.PizzaID;  
```  
This query outputs the order details, including customer name, pizza name, quantity, and subtotal. The data can be exported for further analysis or reporting purposes.

## Securing SQL Scripts Against Injection

SQL injection is a significant security concern for database systems. To mitigate this risk, the Pizza Takeaway Database implements the following measures:  
  
1. \*Prepared Statements:\*  
- Parameterized queries separate SQL commands from user input, preventing malicious SQL code from being executed.  
Example:  
```sql  
PREPARE stmt FROM 'INSERT INTO Orders (CustomerID, OrderDate, TotalAmount) VALUES (?, ?, ?)';  
EXECUTE stmt USING @customer\_id, @order\_date, @total\_amount;  
```  
  
2. \*Input Validation:\*  
- User inputs are validated and sanitized to ensure they conform to expected formats and data types.  
  
3. \*User Privileges:\*  
- Database users are granted only the permissions necessary to perform their tasks, minimizing the impact of potential attacks.

## GitHub Repository

The complete project, including SQL scripts, the ERD diagram, and this report, is hosted on GitHub. The repository contains:  
- SQL scripts for table creation and data insertion.  
- The ERD diagram (as a doc file).  
- A README file with instructions for setting up and running the database.  
  
Access the repository via the following link:  
[GitHub Repository Link]