<Online Shop>

Milestone 1: Requirements Analysis & Conceptual Design

Description:
This system tries to implement an online shop.
Online shop has different customers, for every customer User Id, first name, last name and Telephone number is stored. A customer can be a prime or normal member. Customers can order different products available and for every order , id, User id, date and product id should be saved. For every Product , Id, name, Category, Brand and Price is stored.
Shop has different Employees that have an Id , first name , last name and Telephone Number.
Our shop has access to different suppliers that have Id, name and Telephone number.
A delivery happens with three parties involved : Employee, Product and supplier.
For every delivery the date is stored.
An Employee can supervise other Employees

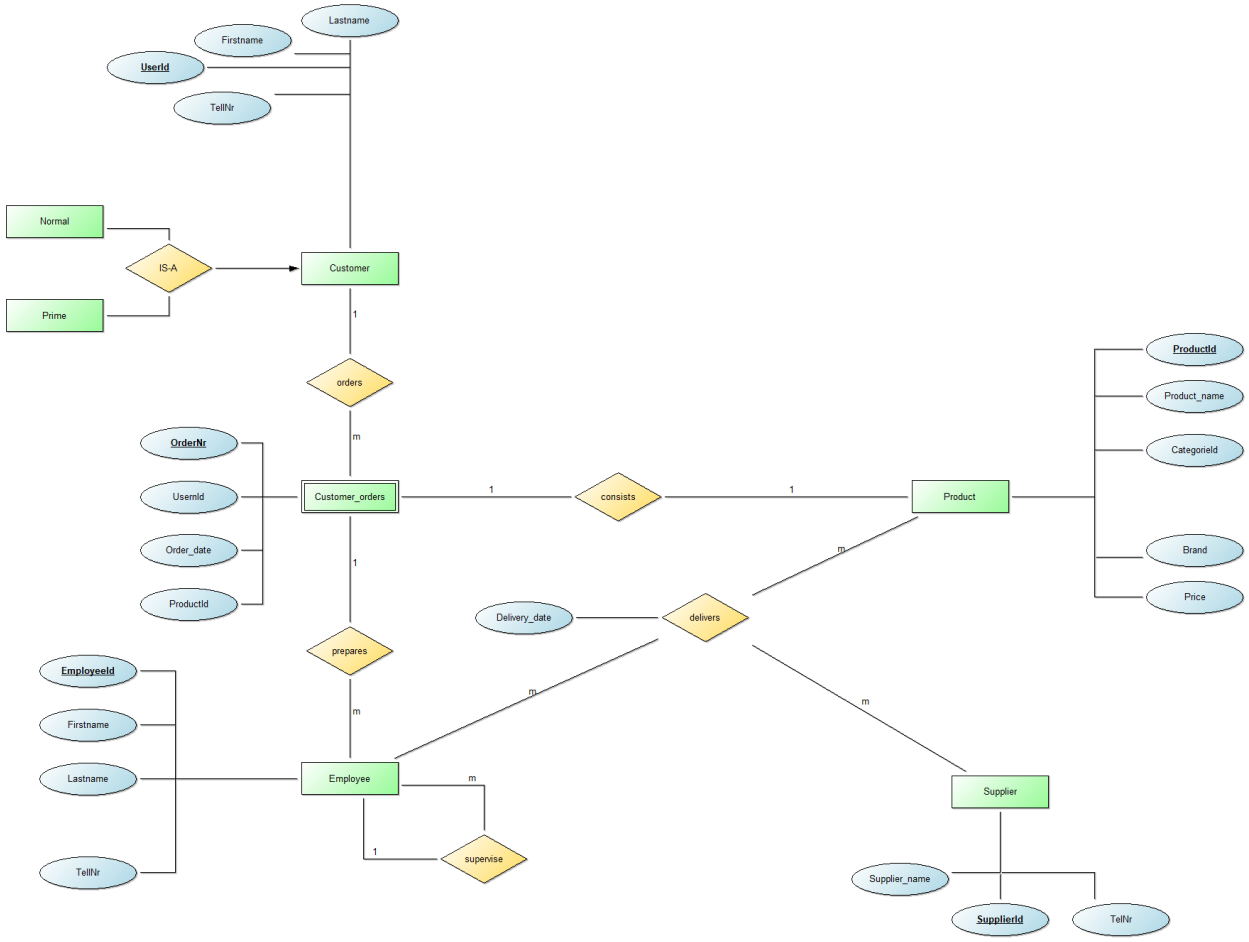

Figure 1: Entity Relationship Diagram

Milestone 2: Logical Desgin

relation1 (attr1, attr2, ..., *attrN*)
PK: attr1
FK: attrN ◊ relationN
Candidate Keys:

Customer(UserId, Firsname, Lastname, TelNr)
PK:UserId

Prime(UserId)
PK:UserId
FK: Prime.UserId ◊ Customer

Normal(UserId)
PK:UserId
FK: Normal.UserId ◊ Customer

Customer_orders(OrderNr, UserId, ProductId, Order_date)
PK:OrderNr
FK: Customer_orders.UserId ◊ Customer

Employee(EmployeeId, Firstname, Lastname, TelNr)
PK:EmlpoyeeId

Supervise(EmployeeId)
PK:EmployeeId
FK: supervise.EmployeeId ◊ Employee

Product(ProductId, Product_name, Brand, CategorieId, Price)
PK:ProductId

Supplier(SupplierId, Supplier_name, TelNr)
PK:SupplierId

delivers(EmployeeId, SupplierId, ProductId)
PK: EmployeeId, SupplierId, ProductId
FK:delivers.EmployeeId◊Employee,delivers.SupplierId◊Supplier,delivers.ProductId◊Product

Milestone 4: Implementation

Java

First a connection to the data bank should be opened, this could be done with the help of jdbc libraries. Then different string arrays with different values for name, last name, etc… is implemented. Then every table is filled with random values from the pre defined arrays. This could simply be copied for every table that doesn't contain any foreign keys. The Ids for table entries should be unique but because we have triggers in database for every id, they are automatically unique and we shouldn't have any concerns about that.

The Process for the tables containing foreign keys are a little bit more tricky. I decided to get the keys needed to fill the new table with a "Select neededId from neededtable" and fill an array list with these keys. Then simply using a for each array on these array list, the tables with need of these keys could be filled.

At the end we should count the rows of our tables and output them.

Then the connection should be closed and we are done.

```
public class TestDataGenerator {
    public static void main(String[] args)throws ClassNotFoundException, SQLException {
        //connect info
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String database = "jdbc:oracle:thin:@oracle-lab.cs.univie.ac.at:1521:lab";
        String user = "a01428941";
        String pass = "dbs19";
        Connection con = DriverManager.getConnection(database , user, pass);
        Statement stmt = con.createStatement();
        //connection to Oracle Database Started

        String[] Firstname = {"'Max'","'Philipp'","'Luke'","'John'","'Mohammad'","'Ali'",
                "'Lionel'","'Cristiano'","'Luis'","'Natalia'","'Gabriel'","'Stefanie'",
                "'Mia'","'Selena'","'Nicole'","'Nick'","'Ellie'","'Emma'","'Olivia'","'Sophia'"};
        String[] Lastname = {"'Smith'","'Jones'","'Williams'","'Brown'","'Davis'","'Miller'",
                "'Willson'","'Schneider'","'Fischer'","'Weber'","'Wagner'","'Meyer'",
                "'Koch'","'Richter'","'Klein'","'Beck'","'Huber'","'Fuchs'","'Jung'","'Rivera'"};
        String[] Product_name= {"'p1'","'p2'","'p3'","'p4'","'p5'","'p6'","'p7'","'p8'","'p9'","'p10'"
                ,"'p11'","'p12'","'p13'","'p14'","'p15'","'p16'","'p17'","'p18'","'p19'","'p20'"};
        String[] Brand= {"'Samsung'","'Apple'","'Sony'","'Huawei'","'Microsoft'","'Lenovo'","'Asus'"
                ,"'Panasonic'","'Nintendo'","'HTC'"};
        String[] CategorieId= {"'Smart Phone'","'Smart Watch'","'TV'","'Console'","'Laptop'","'Speaker'"
                ,"'Tablet'","'VR'","'Gift Card'","'Printer'"};
        String[] Supplier_name= {"'Saturn'","'Mediamarkt'","'Amazon'","'Digikala'","'Apple Store'","'T-Mobile'"
                ,"'A1'","'3'","'Hofer'","'Libro'"};
```

```
Problems  Javadoc  Declaration  Console ×
<terminated> TestDataGenerator [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Jun 20, 2019, 11:55:54 AM)
Number of datasets Product: 100
Number of datasets Employee: 100
Number of datasets Supplier: 100
Number of datasets prime: 1000
Number of datasets delivers: 1000
Verbindung Ende
```

PHP

Firstly a database helper is implemented to connect the program to database. The main page of site uses this class to connect to database(This could also be done Implicitly in the main page, I only thought that this way it looked cleaner)

```php
<?php

// Include DatabaseHelper.php file
require_once('DatabaseHelper.php');

// Instantiate DatabaseHelper class
$database = new DatabaseHelper();

?>
```

Then different functions using needed parameters for class databaseHelper is implemented such as:
Adding a customer

```php
public function insertIntoCustomer($firstname, $lastname, $telNr)
{
    $sql = "INSERT INTO Customer (Firstname, Lastname, TelNr) VALUES ('{$firstname}', '{$lastname}', '{$telNr}')";

    $statement = @oci_parse($this->conn, $sql);
    $success = @oci_execute($statement) && @oci_commit($this->conn);
    @oci_free_statement($statement);
    return $success;
}
```

This function should be called in the index file and the buttons and entries for parameters should be implemented

```html
    <h2>Add Customer: </h2>
    <form method="post" action="addCustomer.php">
        <div>
            <label for="new_firstname">Firstname:</label>
            <input id="new_firstname" name="firstname" type="text" maxlength="20">
        </div>
        <br>
        <div>
            <label for="new_lastname">Lastname:</label>
            <input id="new_lastname" name="lastname" type="text" maxlength="20">
        </div>
        <br>
        <div>
            <label for="new_telNr">TelNr:</label>
            <input id="new_telNr" name="telNr" type="number" max="9999999999">
        </div>
        <br>
        <div>
            <button type="submit">
                Add Customer
            </button>
        </div>
    </form>
```

This part of html posts these variables to another php file connected to server in order to call the desired function and also preventing errors in hompage.

index.php | addCustomer.php | customerOrders.php

```php
    $lastname = $_POST['lastname'];
}

$telNr = '';
if(isset($_POST['telNr'])){
    $telNr = $_POST['telNr'];
}

// Insert method
$success = $database->insertIntoCustomer($firstname, $lastname, $telNr);

// Check result
if ($success){
    echo "Customer '{$firstname} {$lastname}' successfully added!'";
}
else{
    echo "Error can't insert Person '{$firstname} {$lastname}'!";
}
?>

<!-- Link back to index page-->
<br>
<a href="index.php">
    go back
</a>
```

At the end a link to the homepage is added so that user could go back to homepage after either success or error of the function.
This method is simply repeated for other functions like : delete customer, update customer, add product.

The following screenshot explains the functionality of searching for specific categories in the online shop.

```php
<div>
    <form id='searchform' action='index.php' method='get'>
        Search product for categorie:
        <input id='search' name='search' type='text' size='20' placeholder="categorie"
               value='<?php isset($_GET['search']) ? $_GET['search'] : null; ?>'/>
        <input id='submit' type='submit' value='Search'/><br>
    </form>
</div>
<?php
if (isset($_GET['search'])) {
    $sql = "SELECT productid, product_name, brand, price FROM Product WHERE categorieId like '%" . $_GET['search'] . "%'";
    echo "<td>" . $_GET['search'] . "</td>";
    $stmt = oci_parse($database->conn, $sql);
    oci_execute($stmt);
    ?>
    <table style='border: 1px solid #DDDDDD'>
        <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Brand</th>
            <th>Price</th>
        </tr>
        </thead>
        <tbody>
        <?php
        // fetch rows of the executed sql query
        while ($row = oci_fetch_assoc($stmt)) {
            echo "<tr>";
            echo "<td>" . $row['PRODUCTID'] . "</td>";
            echo "<td>" . $row['PRODUCT_NAME'] . "</td>";
            echo "<td>" . $row['BRAND'] . "</td>";
            echo "<td>" . $row['PRICE'] . "</td>";
            echo "</tr>";
        }
        ?>
        </tbody>
    </table>
    <div>A total of <?php echo oci_num_rows($stmt); ?> datasets found!</div><br>
    <?php
    oci_free_statement($stmt);
}
?>
```

For example searching “Smart Phone” will deliver a list containing all smart phones in products.

Search product for categorie: categorie Search

Smart Phone

ID	Name	Brand	Price
1	S10	Samsung	799
2	S10+	Samsung	899
3	X	Apple	899
4	XS	Apple	999

A total of 4 datasets found!

Then functionality of adding orders using User Id and Product Id is added and to separate links to show complete list of orders and customers are implemented(this sends the user to a different page And at the end of the page a link to go back to homepage is implemented)

The challenge of showing order list is that it contains a foreign key referencing to customer ids. In order to display the name of the person requesting the order, we should look for the foreign key in its original table (Customer here).This could be done using a second SQL select function.

```
33      while ($row = oci_fetch_assoc($stmt)) {
34          //Selecting FOREIGN key of order to get names
35          $sqlHelp = "SELECT Firstname, Lastname FROM customer where UserId like '%".$row['USERID']."%' ";
36          $stmtHelp = oci_parse($conn, $sqlHelp);
37          oci_execute($stmtHelp);
38          $rowHelp = oci_fetch_assoc($stmtHelp);
39
40
41          echo "<tr>";
42          echo "<td>" . $row['ORDERNR'] . "</td>";
43          echo "<td>" . $rowHelp['FIRSTNAME'] ." ". $rowHelp['LASTNAME'] . "</td>";
44          echo "<td>" . $row['PRODUCTID'] . "</td>";
45          echo "<td>" . $row['ORDER_DATE'] . "</td>";
46          echo "</tr>";
47
48          oci_free_statement($stmtHelp);
49      }
50      ?>
```

A stored procedure with one parameter(number) is created in database and is used to change the price of every product with the given amount (This could be used for example during sales in order to lower the prices of products)

```
97  public function price_change($n)
98  {
99      $sql ="BEGIN price_change('{$n}'); END;";
100     $statement = @oci_parse($this->conn, $sql);
101     $success = @oci_execute($statement) && @oci_commit($this->conn);
102     @oci_free_statement($statement);
103     return $success;
104 }
```