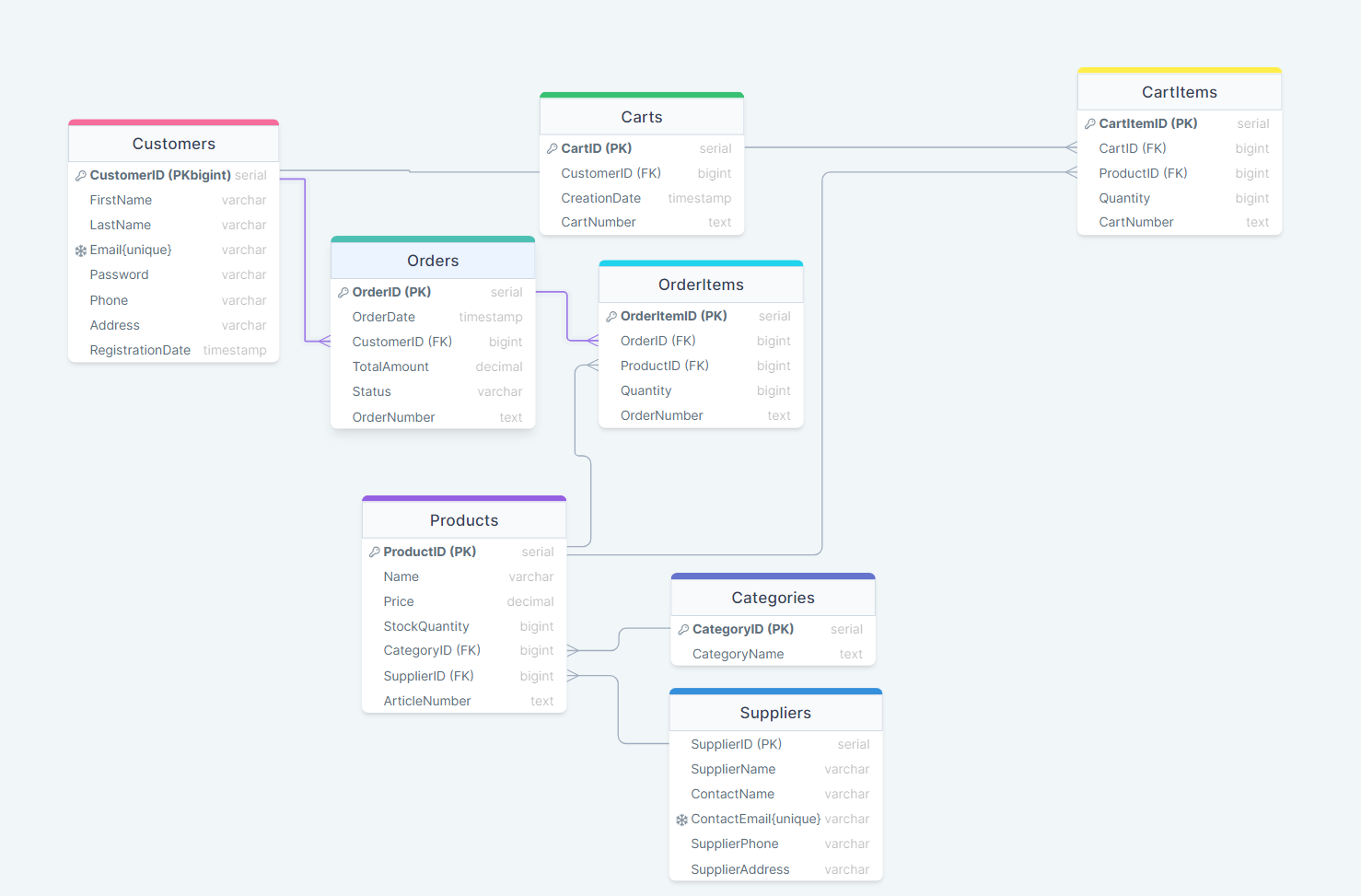
# Jewelry Online store

1. Introduction:

This document provides an overview of the creation of a database system tailored for an e-commerce application. It elaborates on the steps involved in designing and deploying the essential database objects that underpin the functionality of the application. The report discusses the formulation of an ER diagram, the creation of an OLTP solution, and the organization of data for integration into the database. It also covers the establishment of an OLAP solution, which includes a multidimensional data warehouse equipped with fact tables and Slowly Changing Dimensions (SCD). Furthermore, the report delineates the design of an Extract, Transform, Load (ETL) process for migrating data.[ER-diagram](https://drawsql.app/teams/uladzislau-bandarenka/diagrams/jewelrystoredb):



1. OLTP schema description  
   Database Tables:

**Customers Table**

* **Purpose:** Stores information about customers.
* **Columns:**
  + CustomerID: Primary key, unique identifier for each customer.
  + FirstName: Customer's first name.
  + LastName: Customer's last name.
  + Email: Customer's email address, unique.
  + Password: Customer's password.
  + Phone: Customer's phone number.
  + Address: Customer's address.
  + RegistrationDate: Timestamp of customer registration.

**Categories Table**

* **Purpose:** Stores product categories.
* **Columns:**
  + CategoryID: Primary key, unique identifier for each category.
  + CategoryName: Name of the category, unique.

**Suppliers Table**

* **Purpose:** Stores information about suppliers.
* **Columns:**
  + SupplierID: Primary key, unique identifier for each supplier.
  + SupplierName: Name of the supplier.
  + ContactEmail: Email address of the supplier contact, unique.
  + SupplierPhone: Phone number of the supplier.
  + SupplierAddress: Address of the supplier.

**Products Table**

* **Purpose:** Stores information about products.
* **Columns:**
  + ProductID: Primary key, unique identifier for each product.
  + ProductName: Name of the product.
  + Price: Price of the product.
  + StockQuantity: Quantity of the product in stock.
  + ArticleNumber: Unique article number for the product.
  + CategoryID: Foreign key referencing CategoryID in the Categories table.
  + SupplierID: Foreign key referencing SupplierID in the Suppliers table.

**Carts Table**

* **Purpose:** Stores information about customer carts.
* **Columns:**
  + CartID: Primary key, unique identifier for each cart.
  + CustomerID: Foreign key referencing CustomerID in the Customers table.
  + CreationDate: Timestamp of cart creation.
  + CartNumber: Unique identifier for the cart.

**CartItems Table**

* **Purpose:** Stores information about items in customer carts.
* **Columns:**
  + CartItemID: Primary key, unique identifier for each cart item.
  + CartID: Foreign key referencing CartID in the Carts table.
  + ProductID: Foreign key referencing ProductID in the Products table.
  + Quantity: Quantity of the product in the cart.
  + CartNumber: Cart number associated with the item.

**Orders Table**

* **Purpose:** Stores information about customer orders.
* **Columns:**
  + OrderID: Primary key, unique identifier for each order.
  + OrderDate: Timestamp of the order.
  + CustomerID: Foreign key referencing CustomerID in the Customers table.
  + TotalAmount: Total amount of the order.
  + Status: Status of the order (e.g., pending, completed).
  + OrderNumber: Unique identifier for the order.

**OrderItems Table**

* **Purpose:** Stores information about items in customer orders.
* **Columns:**
  + OrderItemID: Primary key, unique identifier for each order item.
  + OrderID: Foreign key referencing OrderID in the Orders table.
  + ProductID: Foreign key referencing ProductID in the Products table.
  + Quantity: Quantity of the product in the order.
  + OrderNumber: Order number associated with the item.

**Relationships and Constraints:**

1. **Foreign Key Constraints:** Ensure referential integrity between related tables (e.g., Products references Categories and Suppliers, Carts references Customers, etc.).
2. **Unique Constraints:** Enforce uniqueness for certain columns like Email, CategoryName, ContactEmail, ArticleNumber, CartNumber, and OrderNumber.

### Indexes:

**Index on CartItems**

**Index Name:** idx\_cart\_items

**Table:** CartItems

**Columns Included:** ProductID, CartNumber

**Type:** Unique Index

**Purpose:**

* The unique index on CartItems ensures that each combination of ProductID and CartNumber is unique within the table. This means that for a given cart, a specific product can only appear once.
* This index helps to enforce data integrity by preventing duplicate entries of the same product within a single cart.
* The index can also improve query performance when searching for items within carts, especially when queries involve both ProductID and CartNumber.

**Benefits of Unique Indexes:**

1. **Data Integrity:** Unique indexes ensure that the combination of columns specified in the index is unique across the table, preventing duplicate entries.
2. **Performance Optimization:** Indexes improve the speed of data retrieval operations by allowing the database to find rows more quickly and efficiently.
3. **Enforcing Business Rules:** By enforcing uniqueness on combinations of columns, these indexes help maintain the integrity of the data in line with business requirements.

**Potential Considerations:**

1. **Storage Overhead:** Indexes require additional storage space, so be mindful of the trade-off between improved query performance and the storage overhead.
2. **Insert/Update Performance:** While indexes speed up read operations, they can slow down insert and update operations because the index must be maintained.

In summary, these unique indexes are well-designed to enforce data integrity and optimize query performance for operations involving ProductID and OrderNumber in the OrderItems table and ProductID and CartNumber in the CartItems table.

Top of Form

Bottom of Form

### Key Functions and Procedures:

**Function: AddProductToCart**

**Function Name:** AddProductToCart

**Parameters:**

* cartID (BIGINT): The ID of the cart to which the product is being added.
* productID (BIGINT): The ID of the product being added to the cart.
* quantity (BIGINT): The quantity of the product to add to the cart.

**Returns:** VOID (This function does not return any value)

**Purpose:**

* This function adds a product to a specified cart. If the product already exists in the cart, it updates the quantity by adding the new quantity to the existing quantity.

**Function Logic:**

1. Attempts to insert a new row into the CartItems table with the specified cartID, productID, and quantity.
2. If a conflict occurs (i.e., the product already exists in the cart), the function updates the quantity of the existing product by adding the new quantity to the current quantity.
3. Data Loading and Transformation

**Function: PlaceOrder**

**Function Name:** PlaceOrder

**Parameters:**

* customerID (BIGINT): The ID of the customer placing the order.

**Returns:** BIGINT (Returns the ID of the newly created order)

**Purpose:**

* This function handles the process of placing an order for a customer. It creates a new order, transfers items from the customer's cart to the order, calculates the total amount, and clears the cart.

**Function Logic:**

1. Inserts a new row into the Orders table with the specified customerID, an initial TotalAmount of 0, and a Status of 'Pending'. The newly created OrderID is stored in newOrderID.
2. Inserts rows into the OrderItems table by selecting all items from the customer's cart, joining with the Products table to get the price, and associating them with the new order.
3. Updates the TotalAmount in the Orders table by calculating the sum of the quantities multiplied by the unit prices from the OrderItems table for the new order.
4. Deletes all items from the customer's cart.
5. Returns the OrderID of the newly created order.

**How to upload csv files**

To upload csv files to the database, you need to install PyCharm, and you also need to create a project and upload all files with the extension .py and create a separate folder in the project for csv files and upload them there from Github for convenience, name the folder with csv files "files" in PyCharm, you need to link your PostgreSQL with files with the extension .py must save your password and username, as well as the database name. It is necessary to write the csv files in the following order: customers.py together with customers.csv, products.py with products.csv, orders.py with orders.csv, carts.py with carts.csv.

You need to run projects in the current file mode.