**A PROJECT DOCUMENTATION**

**on**

***ORDER PROCESSING SYSTEM***

***By***

***WFS PUNE 5 TEAM 2***

**ABSTRACT**

The B2B world operates differently from the B2C or C2C world. Businesses work with other businesses on credit. When a buyer business orders goods from the seller business, the seller business issues an order invoice for the same. This invoice for the goods contains various information like the details of the goods purchased and when it should be paid.

So in order to deal with this problem statement, we have come up with Order Processing System. It is a system which will take the necessary data from the database and generate invoices. This task will run every at midnight and generates the invoices for all orders approved till the day prior. This system does not generate a new invoice but, it updates the database.

**Tech Stack - Java, SQL, Html, Css, JavaScript**

**FRONTEND:**

**ORDER MANAGEMENT PAGE**

### **1. Introduction**

This documentation provides an overview of the frontend components and functionality of the "Order Management" project. The project is designed to manage orders, including quotes pending for approval and approved/completed orders.

### **2. Project Structure**

* **HTML**: The main HTML file is index.html, and external styles are linked from main.css.
* **JavaScript**: The JavaScript logic is contained in app.js.
* **Data**: JSON data is stored within the JavaScript file.

### **3. HTML Structure**

#### **Navigation Bar**

* The navigation bar (<div class="topnav">) provides links to the home page, order management, and login page.

#### **Quotes Pending for Approval Section**

* Displays a header for "Quotes Pending for Approval."
* Contains buttons to import data and manage quotes.
* An unordered list (<ul id="quoteList">) is used to display quote items dynamically.

#### **Approved/Completed Orders Section**

* Displays a header for "Approved/Completed Orders."
* An unordered list (<ul id="orderList">) is used to display order items dynamically.

### **4. JavaScript Functionality**

#### **JSON Data**

* JSON data for quotes and orders is stored within the jsonData object.
* Quotes and orders have properties such as ID, date, shipping cost, total value, and status.

#### **Data Fetching**

* The fetchData function is used to asynchronously fetch JSON data from an external source.

#### **Rendering Functions**

* renderQuoteItems and renderOrderItems dynamically populate the quote and order lists with data.
* approveQuote handles quote approval and status updates.
* showQuoteDetails and showInvoice functions are placeholders for additional functionality.
* isSameDay checks if two dates are the same day.

#### **Initial Rendering**

* The initial rendering of the page is done by calling the rendering functions with JSON data.

### **5. Data Handling**

* JSON data is used to store and manipulate quotes and orders.
* Data is fetched asynchronously from an external source using the fetchData function.
* Data is displayed in the HTML dynamically using rendering functions.

### **6. User Interface**

* The user interface includes navigation links, buttons for actions, and lists to display data.
* Buttons trigger actions like importing data, approving quotes, and showing details.
* The user interface is designed to be intuitive and responsive.

**BACKEND:**

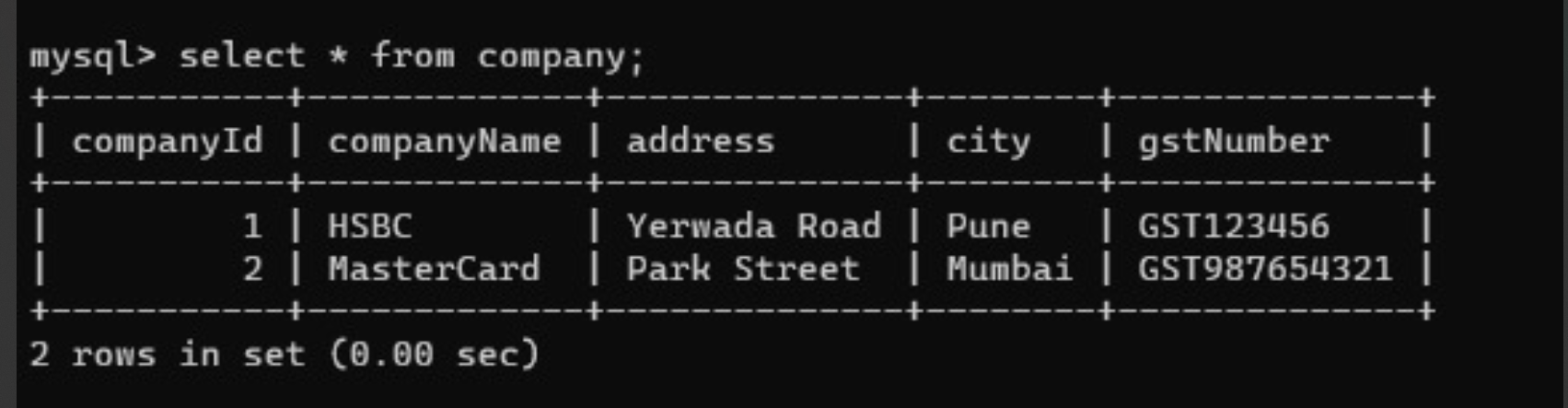
We have approved the business problem in an objective manner and have implemented the MVC architecture (Model View Controller) to solve it. We have created the following layers for easier interpretation.

1. **Domain layer** - The Domain layer represents the core business logic of our application. It typically consists of domain objects or entities that model the real-world concepts our application deals with (e.g., Customer, Invoice, Order). These objects encapsulate data and behavior related to those concepts. We have made Pojo’s for all tables in our database.
2. **DAO (Data Access Objects)** **layer**- The DAO layer is responsible for data access and persistence. It provides interfaces and classes for performing CRUD operations on the database or any other data source. These classes interact directly with the database, hiding the underlying data access complexity from higher layers.
3. **Services layer -** The Services layer contains business logic and services that operate on domain objects. It acts as an intermediary between the domain layer and the DAO layer, coordinating interactions and implementing higher-level functionality. It also deals with Exception handling and interfaces.
4. **Main layer -** This is the Main class with the main method to call all the layers in the correct sequential order.

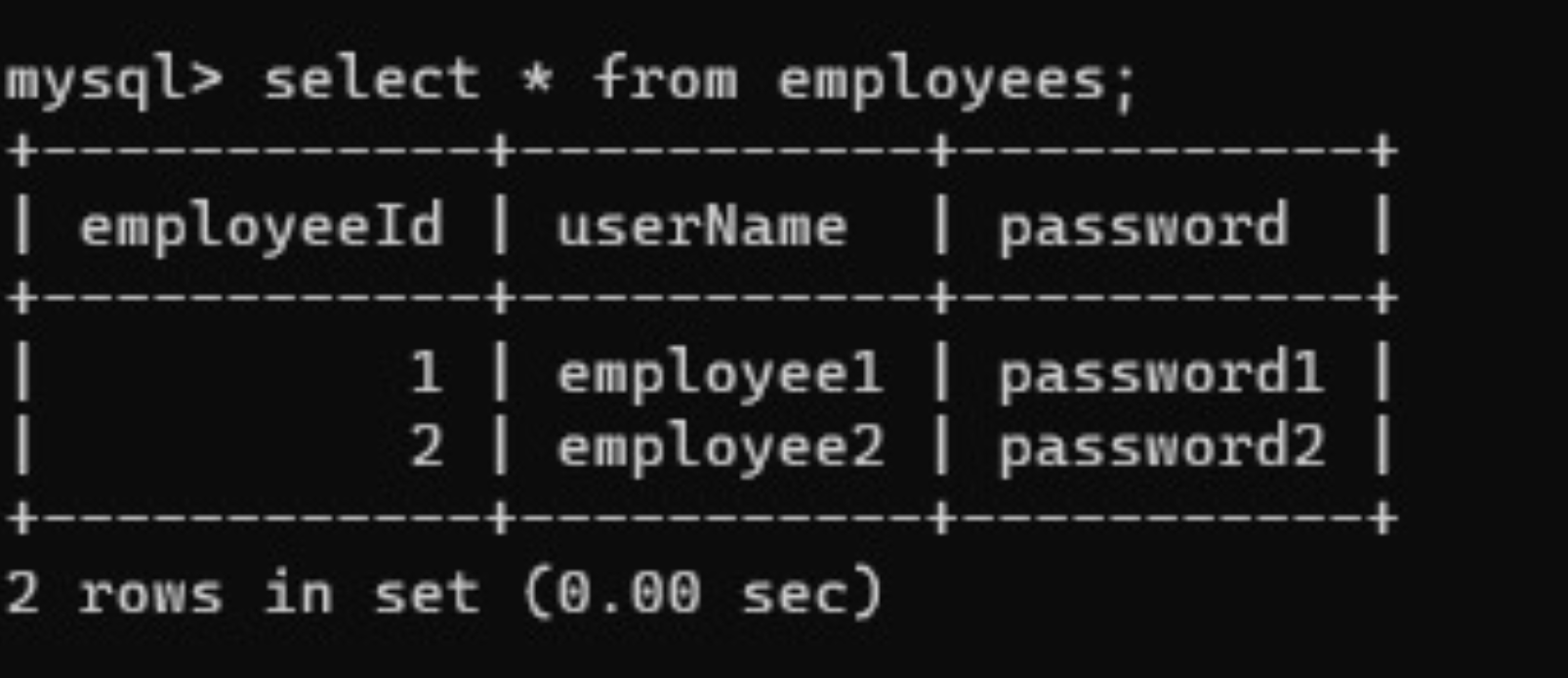
**DATABASE:**

We have created the following tables in our database for the order management Process, with relationships between them.

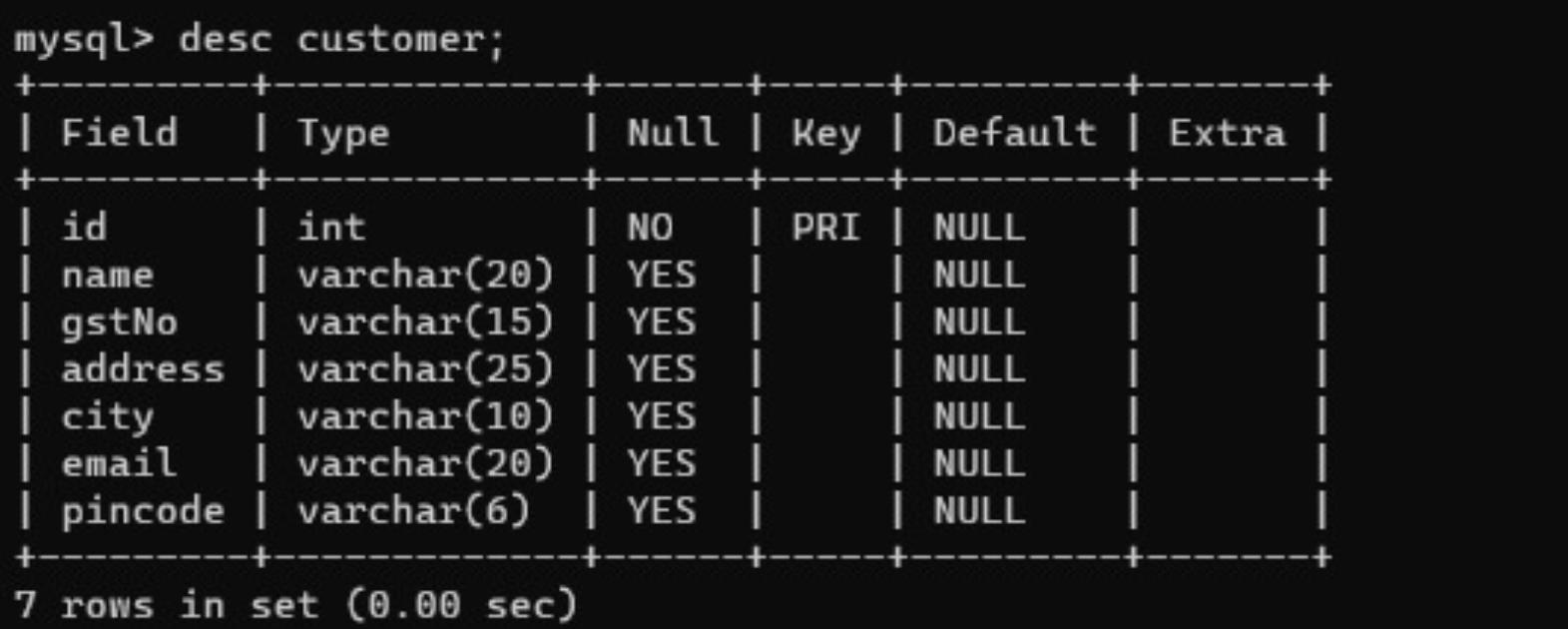
**1.Company**



**2. Employee**

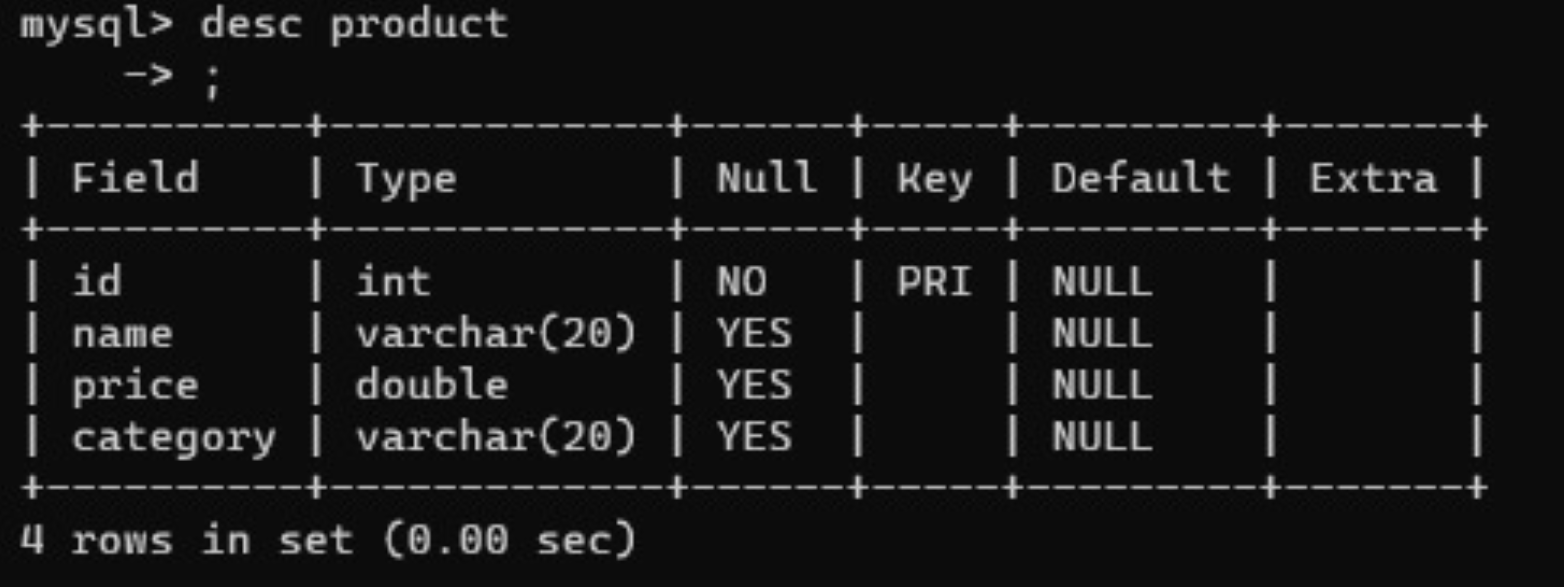
****

**3. Customer**

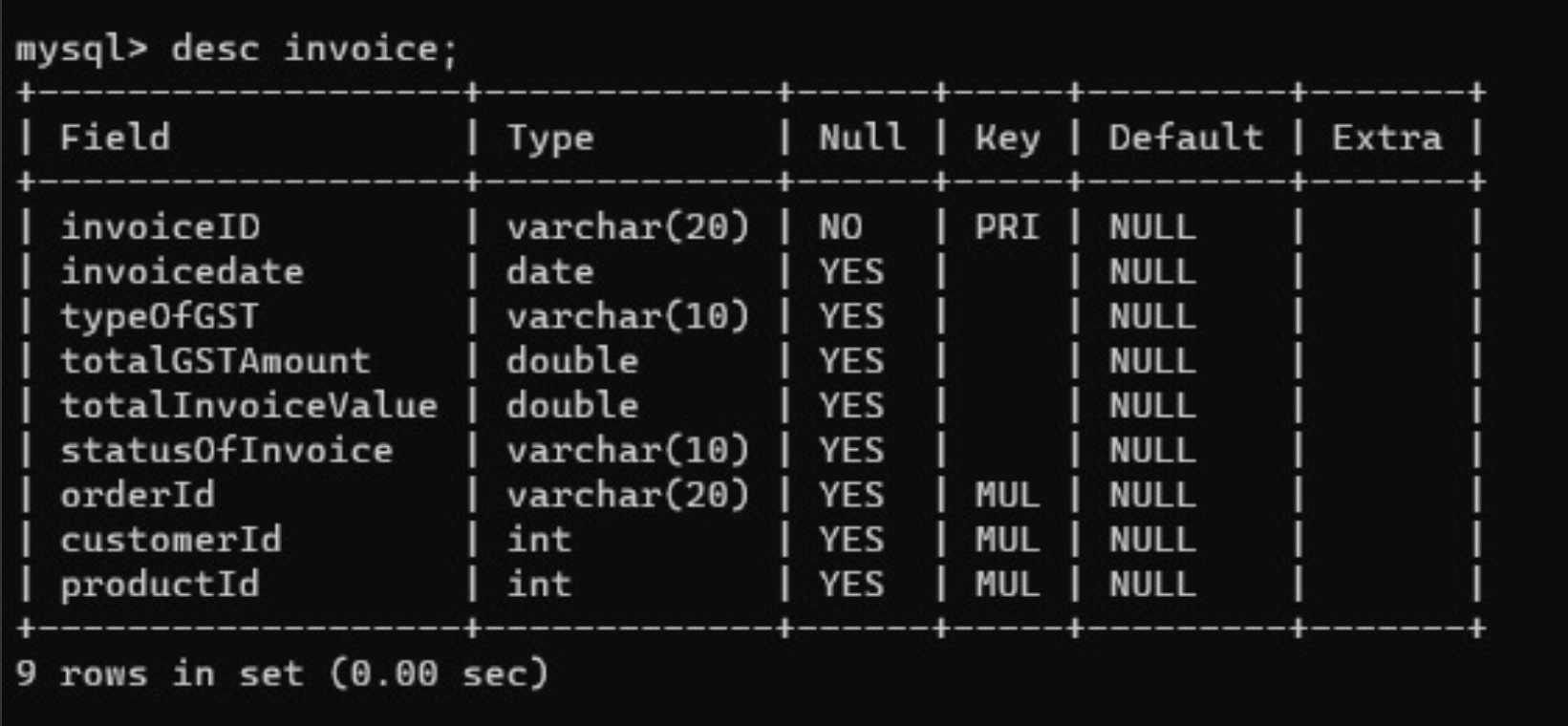
****

****

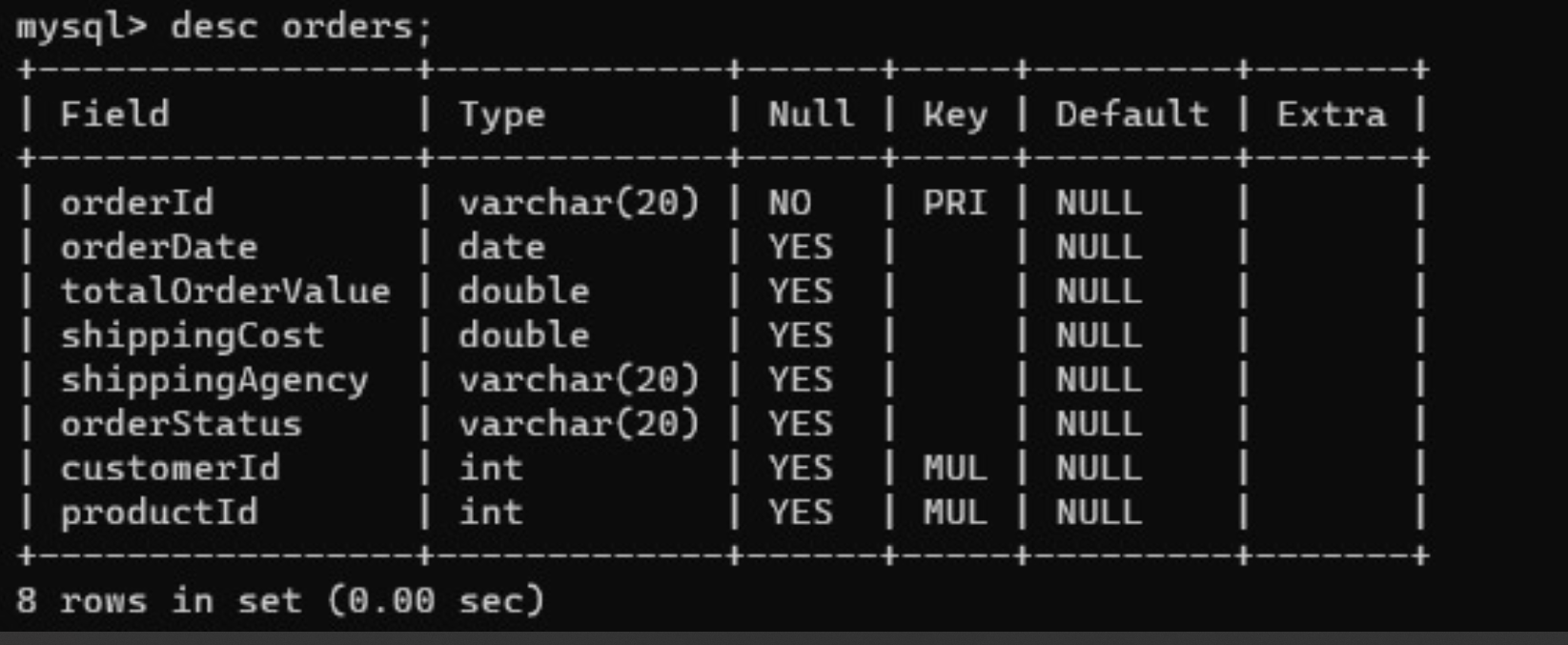
**4. Product**

****

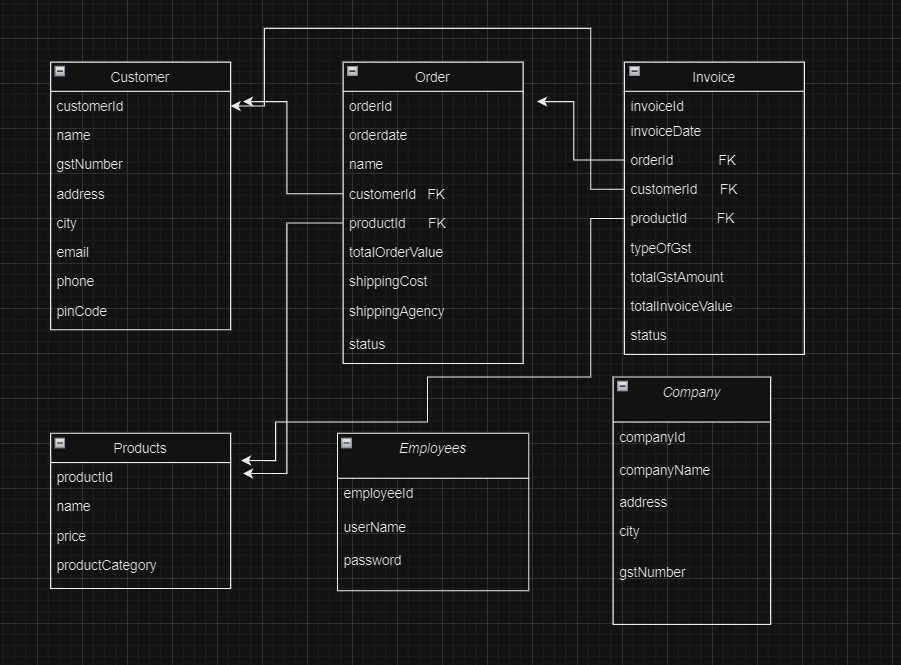
**5. Invoice**

****

**6. Orders**

****

**Entity Relationship Diagram-**

****