JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Bent Spoke Bike Company Database

Course Section: CS605.641.81
Fall, 2022

Prepared by

Sam Dreyfuss
11/19/2022

Table of Contents

1. Introduction

1.1.Scope and Purpose of Document

The scope and purpose of this document is to facilitate the building of an online system for the Bent Spoke Bike Store which sells millions of bikes yearly (and growing) across the United States. Business is currently managing manually via spreadsheets, emails and phone calls. Due to the yearly growth, it will soon no longer sustain current process and would now like to explore the use of an online application. The Bent Spoke Bike Store has international market aspirations and ensure data across the company remains up-to-date and accurate will be paramount for the company to each its full potential. This document will serve as a written record to provide all levels of detail pertaining but not limited to the conceptual design, logical design and implementation details of the company's new database system.

1.2.Project Objective

The objective of this project is to first do the database design that will capture Inventory and Sales as well Reporting for our Sales team, Marketing and Finance. The first phase is to do data conversion of the various spreadsheets across all the stores throughout the United States into our new database so that reporting and dashboards can be utilized. The second phase will be to create the UI/application that will interact with the database. By improving the company's data storage infrastructure, the company will be more organized compared to competitors which will result in improved business operations leadings to higher customer satisfaction and retention. Higher customer satisfaction will result in improved company profitability and a more enjoyable work experience for all employees. As the database implementation proves successful

2. System Requirements

2.1 Hardware Requirements

As the bike store will be modelling will be a US focused bike store looking to dominate the US bicycle retail market driven by our competitive prices, I am planning for scale accordingly. There are approximately 50 mill cyclists in the US, therefore I planned for the possibility of 50 million customers.

To ensure the hardware can handle up to 50 million customers, I decided to use hardware for a large scale (enterprise) system. Including:

1) **Platform:** Windows 2008 R2 (64-bit), Intel Core i7
   CPU: 8 vCPU, Frequency: 2.2GHz and threads :16
2) RAM: 32 Gigabytes

3) **Storage Configuration:** 600 GB (100 GB for the server + 500 GB for the database) 15000 RPM drive or a tier 1 SAN storage (2-4 GBps SAN dedicated channel)

2.2 Software Requirements

Minimum Software Requirements to run applications:

Operating Systems: Windows
Database: MySQL Version 8.0

2.3 Functional Requirements

- The bike store is divided up into different departments which can range from 1 to 5000 people
- Upload spreadsheet (CSV) to Employee table in database (as well as all other tables)
- Allow Employees to create a Customer Account (if the customer doesn't already exist)
- Allow employees to create a sales order
- Keep track of bike inventory the store has on hand
- Keep track special projects (ex. promotional events)
- Keep track of employees and their dependents
- Keep track of customer payments to ensure full payment of bike is received
- Ensure that an increase in product price from wholesaler is reflected in the products inventory record

2.4 Database Requirements

MySQL Version 8.0.30 and MySQL Workbench

3. Database Design Description

3.1 Design Rationale

The design for this database is models off how a bike store's departments, employees, customers, and inventory management interact with different parts of the database and thus different tables. The database design addresses the different types of data and areas that a bike company would need to maintain as well as the projects that an employee might be assigned to, employee information such as their department and company.

Artificial primary keys were used as opposed to an alternative unique ID such as an SSN as this method circumvents the issue of having typos or update anomalies within the primary key - For example if the SSN number was inputted into the database incorrectly or needed to be updated by the employee in the event that they had an their identity stolen and needed to change their SSN. Sequential ID doesn't have the risk of ever having to change from its original value. As a result of only using artificial sequential primary keys, all primary keys were integers.

I chose to show nonidentifying relationships for several entities via solid lines in the ERD if a child's foreign key is part of its primary key. Employee_Company_Project and Product_Wholesaler are represented as identifying relationships in my ERD as they meet the definition. I didn't feel the need to choose non-identifying relationship for two entities when it could be modeled as identifying relationship as there wasn't any tangible benefit to doing so within my data model.

3.2 E/R Model

An Entity Relationship (ER) model is a high-level visualization of a conceptual data model diagram. ER models help to systematically analyze data requirements to produce a well-designed database. The ER Model represents real-world entities and the relationships between them. IE design was chosen due to the benefit of showing the cardinality of every relationship along with an easily interpreted diagram of every entitiy. By using IE notation, data can be well defined and more easily understood relative to other options for ER modeling such as Chen's notation. IE notation becomes the superior choice as the database grows and the complexity increases.

Company_Project – Models the various projects the company is undertaking
Customer – Models the customers who have or are purchasing bikes
Customer_Payment – Models how each customer is paying for their bike
Department – Models the different departments at the bike company
Dependent – Models the dependents associated with employees of the company
Employee – Models the employees who work at the company
Employee_Company_Project – Models which employee is working on which project
Employee_Dependent_Relation – Models the various types of dependent relations there can be at the company
Job_Title – Models the different types of jobs the company offers
Payment_Type – Models the different types of payment types accepted by the company
Product – Models the bikes offered by the company as well as what's available for purchase
Product_Category – Models the different types of bikes offered by the store
Product_Wholesaler – Models the specific bikes being sold to the company by different wholesalers
Sales_Order – Models the sales process during the sale of a bike to a customer
Sales_Order_Detail – Models the more minute details related to each sale
Wholesaler – Models the wholesaler information who supplies bikes to the company

3.2.1 Entities

Employee (Weak entity) – An employee is an individual who works for the bike store selling bikes, helping with inventory management, or ensuring IT is running smoothly. Each employee is identified using a unique employee id which serves as the tables primary key. Each employee's first and last name, job title, date of birth, hire date, primary telephone, supervisor id, and department ids can be found in the employee table. Foreign keys found in the table include job title id, supervisor employee id, and department id. The supervisor employee id also is recursive or self-referenced in the table as it is the primary key

Dependent (Weak entity) – If an employee of the company has a child, they will be listed on the dependent table. Each dependent of an employee is identified using a dependent if (primary key) and also has associated descriptive information including the parents employee id (foreign key), the dependents first name, last name, as well as an employee dependent relation id.

Employee_Dependent_Relation (Strong entity) – Each employee dependent relation can be selected from the employee dependent relation table using a unique id (primary key) as well an associated name for the relation (exp: Daughter, Son).

Job_Title (Strong entity) – Each job title the company offers is housed in the job title entity and is referenced with a job title id (primary key) and job title name.

Department (Strong entity) – Each department at the bike store has its own department id (primary key) as well as an associated department name.

Company_Project (Strong entity) – Each company project being worked on within the company is housed in company project entity and is referenced with a project id (primary key) and a project name.

Payment_Type (Strong entity) – Each payment type accepted at the bike store has its own payment id (primary key) as well as an associated payment type name.

Customer_Payment (Weak entity) – Each customer's payments are kept track of utilizing a unique customer payment id as a primary key in the customer payment table upon purchasing a bike. Each payment also includes the customers id and the payment type id which are both foreign keys in the table.

Sales_Order (Weak entity) – When a bike is purchased, a sales order is generated which includes a unique sales order id, the customers id (FK), the order date, a customer payment id (FK), as well as the employee id who facilitated the sale of the bike (FK).

Sales_Order_Detail (Weak entity) – Sales order detail allows specific details of a given order to be captured via a unique order detail id (primary key). Attributes included on the table also include the sales order id (FK), product id (FK), price of the good, and units sold.

Customer (Strong entity) – A customer is a client of the bike store who has placed an order for a bike. A customer is uniquely identified with a unique customer id which serves as the primary key. Each customer's first name, last name, address, city, state, zip code and their outstanding balance are also stored within the database.

Wholesaler (Strong entity**)** – A wholesaler supplies the bike store with new bikes to sell to customers. Each wholesale is identified with a unique wholesaler id (primary key), as well as the name of the wholesaler, the individual who is the primary contact, the phone number of the contact, and the email of the contact at wholesaler.

Product (Weak entity) – All products sold by the bike store are kept track of in the product table. Each product is identified via a unique product id (primary key) and attributes such as the product category (foreign key), product name, product price, and units in stock.

Product_Category (Strong entity) – All product category types are maintained in the product category table via a unique id (primary key). All product category names are associated with each product category id within the table.

3.2.2 Relationships

- Employee – Job_Title: M:1 (each employee has one job title; a job title can be tied to multiple employees).
    - This is a non-identifying relation
    - Employee (total participation) - Job_Title (total participation)

- Employee – Department:M:1 (each employee works for one department, each department can have multiple employees)
    - This is a non-identifying relation
    - Employee (total participation) – Department (total participation)

- Employee – Company_Project: M:M (each employee can work on multiple company projects; company projects can be worked on by multiple employees)
    - This is an identifying relation as the primary keys from both employee and company project migrate
    - Employee (partial participation) – Company_Project (partial participation)

- Employee – Employee: M:0 (An employee may have a supervisor, a supervisor may have multiple employees)
    - This is a non-identifying relation
    - Employee (partial participation) – Employee(partial participation)

- Employee – Dependent: 1:M (An employee may have multiple dependents, each dependent is only associated with one employee)
    - This is a non-identifying relation
    - Employee (partial participation) – Dependent (Total participation)

- Dependent - Employee_Dependent_Relation : M:1 (Each dependent is associated with one employee dependent relation, an employee dependent relation can be associated with multiple dependents
    - This is a non-identifying relation
    - Dependent (Total participation) - Employee_Dependent_Relation (Total participation)

- Customer_Payment – Payment_Type: M:1 (each customer payment has one payent type, a payment type can be associated with multiple customer payments)
    - This is a non-identifying relation
    - Customer_Payment (Total Participation) – Payment_Type (Total Participation)

- Sales_Order – Customer_Payment: M:1 (Each sales order is associated with one customer payment, each customer payment is associated with one or more sales orders)
    - This is a non-identifying relation
    - Sales_Order (Total Participation) – Customer_Payment (Total Participation)

- Sales_Order – Employee: M:1 (Each sales order is associated with one employee, each each employee can be associated with multiple sales orders)
    - This is a non-identifying relation
    - Sales_Order (Total Participation) – Employee (Partial Participation)

- Customer_Payment – Customer: M:1 (Each Customer payment is associated with one customer, each customer can be associated with one or more sales orders
    - This is a non-identifying relation
    - Customer_Payment (Total Participation) – Customer (Partial Participation)

- Sales_Order – Customer: M:1 (Each sales order is associated with one customer, each customer can be associated with one or more sales orders)
    - This is a non-identifying relation
    - Sales_Order (Total Participation) – Customer (Total Participation)

- Sales_Order_Detail – Sales_Order: M:1 (Each sales order detail is associated with one sales order, each sales order can be associated with one or more orders detail)
    - This is a non-identifying relation

    - Sales_Order_Detail (Total Participation) – Sales_Order (Total Participation)

- Sales_Order_Detail – Product: M:1 (Each sales order detail is associated with one product, each product is associated with one or more sales orders details)
    - This is a non-identifying relation
    - Sales_Order_Detail (Total Participation) – Product (Total Participation)

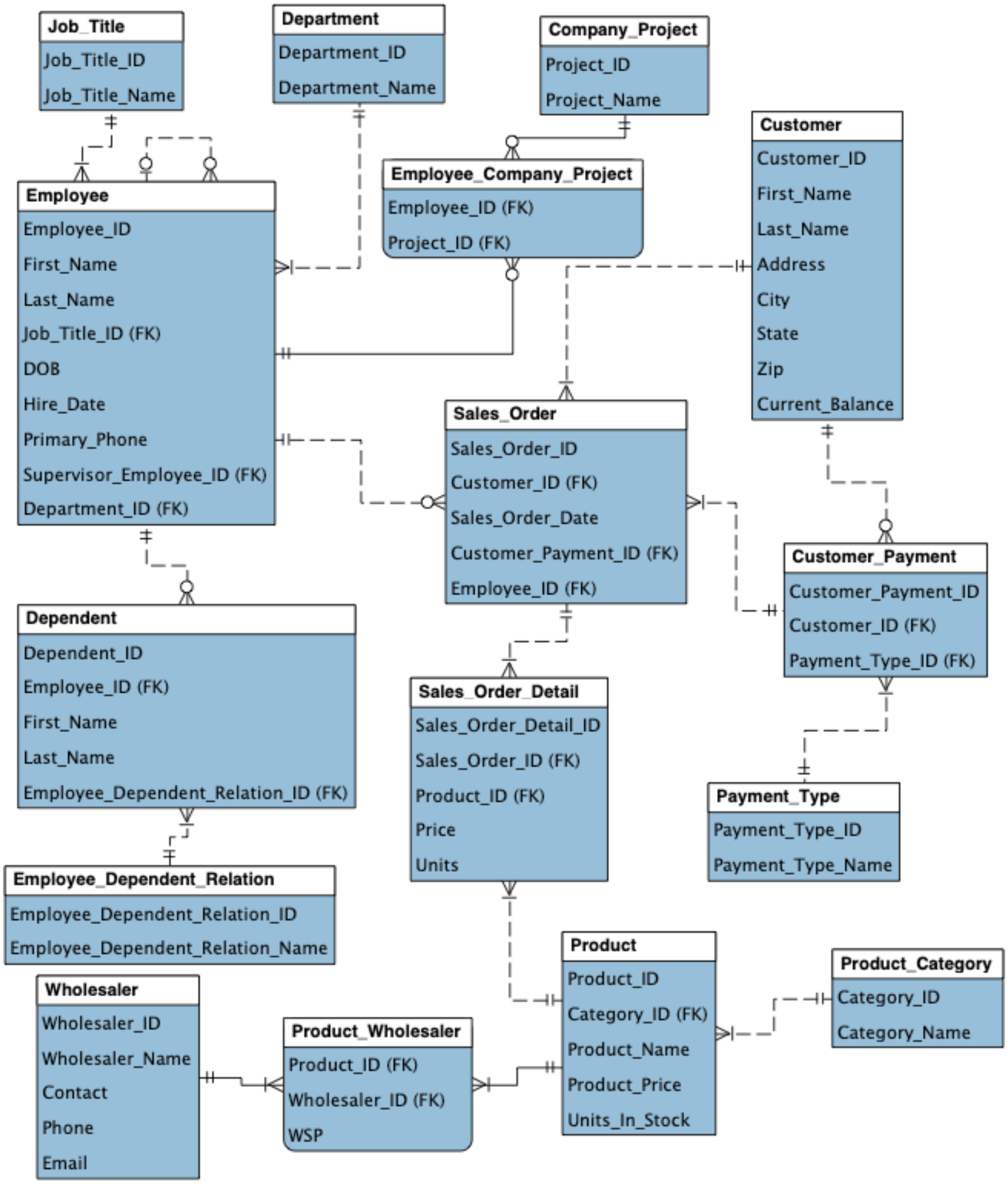
- Product – Product_Category: M:1 (Each product is associated with one product category, each product category is associated with one or more products)
    - This is a non-identifying relation
    - Product (Total Participation) – Product_Category (Total Participation)

- Product_Wholesaler – Wholesaler: M:1 (Each product provided by a wholesaler is associated with one or more wholesalers, each wholesaler is associated with one or more products)
    - This is an identifying relation as there is primary key migration (resulting in composite key)
    - Product_Wholesaler (Total Participation) – Wholesaler (Total Participation)

- Product_Wholesaler – Product: M:1 (Each product provided by a wholesaler is associated with one or more wholesalers, each wholesaler is associated with one or more products)
    - This is an identifying relation as there is primary key migration
    - Product_Wholesaler (Total Participation) – Product (Total Participation)

3.2.3 E/R Diagram

3.3 Relational Model



3.3.1 Data Dictionary

Employee						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Employee_ID	Unique employment id for each employee (autogenerated)	Int	Any	Primary Key	Y	Positive Int
First_Name	First name of employee	Varchar	25	N/A	Y	25 char
Last_Name	Last name of employee	Varchar	25	N/A	Y	25 char
Job_Title_ID	Unique job id for each	Int	Any	Foreign Key	Y	Positive Int
DOB	Data of birth for each employee	Date	8	N/A	Y	‘YYYY-MM-DD’
Hire_Date	Hire date for each employee	Date	8	N/A	Y	‘YYYY-MM-DD’
Primary_Phone	Primary phone number for each employee	Varchar	10	N/A	Y	10 char
Supervisor_Employee_ID	Unique employment id for employee who is also a supervisor	Int	Any	Foreign Key	N	Positive Int
Department_ID	Unique department id for each employee	Int	Any	N/A	Y	Positive Int

Dependent						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Dependent_ID	Unique id for each dependent of an employee (autogenerated)	Int	Any	Primary Key	Y	Positive Int
Employee_ID	Unique id for each company employee	Int	Any	Foreign Key	Y	Positive Int
First_Name	First name of dependent	Varchar	25	N/A	Y	25 Char
Last_Name	Last name of dependent	Varchar	25	N/A	Y	25 Char
Employee_Dependent_Relation_ID	Unique id for each type of employee relation	Int	Any	Foreign Key	Y	Positive Int

Product						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Product_ID	Unique id for each product sold in store (autogenerated)	Int	Any	Primary Key	Y	Positive Int
Category_ID	Unique id for each product category sold in story (autogenerated)	Int	Any	Foreign Key	Y	Positive Int
Product_Name	Name of each product sold by store	Varchar	25	N/A	Y	25 Char
Product_Price	Lists the price of each product	Float	7	N/A	Y	XXXXX.XX
Units_in_Stock	Lists the current total units held in stock	Int	Any	N/A	Y	Positive Int

Wholesaler						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Wholesaler_ID	Unique id for each wholesaler the bike store does business with (autogenerated)	Int	Any	Primary Key	Y	Positive Int
Wholesaler_Name	Name for each wholesaler	Varchar	25	N/A	Y	25 Char
Contact	Name of individual the bike store normally contact at wholesaler	Varchar	25	N/A	Y	25 Char
Phone	Phone number of contact at wholesaler	Varchar	10	N/A	Y	10 Char
Email	Email of contact at wholesaler	Varchar	25	N/A	Y	25 Char

Customer						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Customer_ID	Unique id for each customer of the bike store (autogenerated)	Int	Any	Primary Key	Y	Positive Int
First_Name	First name of customer	Varchar	25	N/A	Y	25 Char
Last_Name	Last name of customer	Varchar	25	N/A	Y	25 Char
Address	Phone number of contact at wholesaler	Varchar	25	N/A	Y	25 Char
City	Email of contact at wholesaler	Varchar	25	N/A	Y	25 Char
State	State of residence for customer	Char	2	N/A	Y	2 Char
Zip	Zip code of customer	Varchar	5	N/A	Y	5 Char
Current_Balance	Most recent monetary balance customer owes to store. Becomes zero once balance has been paid off	Float	7	N/A	Y	XXXXX.XX

Sales_Order

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Sales_Order_ID	Unique sales id for each aggregate sale (autogenerated)	Int	Any	Primary Key	Y	Positive Int
Customer_ID	First name of customer	Int	Any	Foreign Key	Y	Positive Int
Sales_Order_Date	Last name of customer	Date	25	N/A	Y	‘YYYY-MM-DD’
Customer_Payment_ID	Phone number of contact at wholesaler	Int	Any	Foreign Key	Y	Positive Int
Employee_ID	Email of contact at wholesaler	Int	Any	Foreign Key	Y	Positive Int

Customer_Payment

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Sales_Order_ID	Unique sales id for each aggregate sale (autogenerated)	Int	Any	Primary Key	Y	Positive Int
Customer_ID	First name of customer	Int	Any	Foreign Key	Y	25 Char
Sales_Order_Date	Last name of customer	Date	25	N/A	Y	‘YYYY-MM-DD’
Customer_Payment_ID	Phone number of contact at wholesaler	Int	Any	Foreign Key	Y	Positive Int
Employee_ID	Email of contact at wholesaler	Int	Any	Foreign Key	Y	Positive Int

Sales_Order_Detail

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Sales_Order_Detail_ID	Unique sales order id for each item(s) purchased (autogenerated)	Int	Any	Primary Key	Y	Positive Int
Sales_Order_ID	Unique sales id for each aggregate sale (autogenerated)	Int	Any	Foreign Key	Y	Positive Int
Product_ID	Phone number of contact at wholesaler	Int	Any	Foreign Key	Y	10 Char
Price	Email of contact at wholesaler	Float	Any	N/A	Y	XXXXX.XX
Units	Number of unit ordered by customer	Int	Any	N/A	Y	Positive Int

Product_Wholesaler

Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Product_ID	Unique product id for each product wholesaler provides	Int	Any	Primary Key, Foreign Key	Y	Positive Int
Wholesaler_ID	Unique wholesaler id associated with the product providing wholesaler	Int	Any	Primary Key, Foreign Key	Y	25 Char
WSP	Wholesale Price	Float	7	N/A	Y	XXXXX.XX

Employee_Company_Project						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Employee_ID	Unique employee id for each project being worked on	Int	Any	Primary Key, Foreign Key	Y	Positive Int
Project_ID	Unique project id associated with the product being worked on	Int	Any	Primary Key, Foreign Key	Y	Positive Int

Job_Title						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Job_Title_ID	Unique job title id for each type of job the bike store offers	Int	Any	Primary Key	Y	Positive Int
Job_Title_Name	Name of specific title	Varchar	25	N/A	Y	25 Char

Department						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Department_ID	Unique department id for each department found within the bike store	Int	Any	Primary Key	Y	Positive Int
Department_Name	Name of department	Varchar	25	N/A	Y	25 Char

Company_Project						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Project_ID	Unique project id for each project being worked on in the company	Int	Any	Primary Key	Y	Positive Int
Project_Name	Name of project	Varchar	25	N/A	Y	25 Char

Payment_Type						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Payment_Type_ID	Unique payment type id for each payment type accepted by the store	Int	Any	Primary Key	Y	Positive Int
Payment_Type_Name	Payment type name	Varchar	25	N/A	Y	25 Char

Employee_Dependent_Relation						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Employee_Dependent_Relation_ID	Unique dependent relation id for each relation a dependent can have	Int	Any	Primary Key	Y	Positive Int
Employee_Dependent_Relation_Name	Dependent relation name	Varchar	25	N/A	Y	25 Char

Product_Category						
Column Name	Description	Data Type	Size	Constraint Type	Not Null?	Valid Values
Category_ID	Unique product category id for each type of product/bike offered by the store	Int	Any	Primary Key	Y	Positive Int
Category_Name	Product category name	Varchar	25	N/A	Y	25 Char

3.3.2 Integrity Rules

To ensure entity integrity, the database was designed so that every entity had a primary key. To handle the mandatory fields required when creating the tables, the columns were set to NOT NULL. To ensure valid values for each table "look up" tables were used for each drop down option. In order to ensure referential integrity was established foreign key constraints were created. All details regarding integrity rules are listed below. Cardinality ratios are also included for easy reference.

1. **Employees** must have: job title, department, option to have dependents, option to partake in company projects, as well as be included on sales order if they sell a bike

    a. Employee – Job_Title: M:1 (each employee has one job title; a job title can be tied to multiple employees)

    b. Employee – Department:M:1 (each employee works for one department, each department can have multiple employees)

    c. Employee – Company_Project: M:M (each employee can work on multiple company projects; company projects can be worked on by multiple employees)

    d. Employee – Employee: M:0 (An employee may have a supervisor, a supervisor may have multiple employees)

    e. Employee – Dependent: 1:M (An employee may have multiple dependents, each dependent is only associated with one employee)

2. Dependent must have: Associated employee, and employee dependent relation to whom they are related to.

    a. Dependent - Employee_Dependent_Relation : M:1 (Each dependent is associated with one employee dependent relation, an employee dependent relation can be associated with multiple dependents

3. Employee_Company_Project must have: associated employee and project

    a. Employee – Company_Project: M:M (each employee can work on multiple company projects; company projects can be worked on by multiple employees)

4. Customer_Payment must have: Customer and payment type to identify which type of payment a given customer prefers to use.

    a. Customer_Payment – Payment_Type: M:1 (each customer payment has one payent type, a payment type can be associated with multiple customer payments)

    b. Customer_Payment – Customer: M:1 (Each Customer payment is associated with one customer, each customer can be associated with one or more sales orders

5. Sales_Order must have: a customer, customer payment, and employee who helped facilitate the sale.

    a. Sales_Order – Customer_Payment: M:1 (Each sales order is associated with one customer payment, each customer payment is associated with one or more sales orders)

    b. Sales_Order – Employee: M:1 (Each sales order is associated with one employee, each each employee can be associated with multiple sales orders)

    c. Sales_Order – Customer: M:1 (Each sales order is associated with one customer, each customer can be associated with one or more sales orders)

6. Sales_Order_Detail must have: an associated sales order and product id

    - Sales_Order_Detail – Sales_Order: M:1 (Each sales order detail is associated with one sales order, each sales order can be associated with one or more orders detail)

    - Sales_Order_Detail – Sales_Order: M:1 (Each sales order detail is associated with one product, each product is associated with one or more sales orders details)

7. Product must have: a associated category.

    a. Product – Product_Category: M:1 (Each product is associated with one product category, each product category is associated with one or more products)

8. Product_Wholesaler must have: an associated product and wholesaler

    a. Product_Wholesaler – Wholesaler: M:1 (Each product provided by a wholesaler is associated with one or more wholesalers, each wholesaler is associated with one or more products)

    b. Product_Wholesaler – Product: M:1 (Each product provided by a wholesaler is associated with one or more wholesalers, each wholesaler is associated with one or more products)

3.3.3 Operational Rules

Constraint for a few operations include:

1) Company project cannot be deleted if employee is associated

2) We will not be able to delete a department if there's an employee tied to it

3) We will not be able to delete a customer if there's a sales order associated

4) If a wholesaler would like to sell us a new type of bike (for example an electric bike), we must first create the new category of "Electric Bike" within the Product_Catergory table before a new product can be entered into our system.

5) Before a sales order can be saved by an employee, the customers payment type must be selected

3.3.4 Operations

1. Allowing customers to purchase a bike:
    a. Insert: all required data into **sales_order** (sales order id, customer id, sales order date, customer payment id and employee id), **sales_order_detail** (sales order id, sales order detail id, product id, price, and units) and **customer_payment (**customer payment id, customer id, and payment type id)

i.

2. Allowing new customers when they purchase a bike
   a. Insert: all required data into **customers** (customer id, first name, last name, address, city, state, zip, current balance)
3. Employees to sign up for projects
   a. Insert: all required data into **employee_company_project** (employee id, project id)
4. Updated employee information
   a. Update: any updated information which is needed in the **employee** table (employee id, first name, last name, job title id, dob, hire date, primary phone, supervisor employee id, department id.)

3.4 Security

To ensure security of individuals within the database, specifically the customer, employee, and dependent tables – as mentioned previously, I designed the database to not include SSN's. Additionally, I also decided to grant permissions to the select individuals who need to access each table, this will reduce the chance of sensitive information being used for malicious purposes. I implemented these safter features by utilizing the GRANT functionality.

3.5 Database Backup and Recovery

SQL Server has multiple features for database recovery and restoration. A database administrator is allowed restore a set SQL Server backup in a specific order and this allows for a meaningful restoration. A database can be recovered as a whole database, a data file, or a data page which can all be used to attempt to restore a database.

As the bike company cannot afford to lose any inventory data or customer data as this could lead to a slowdown in productivity. The company database will be backed up every 60 min. This will ensure continuity of the business and help to ensure data integrity.

3.6 Using Database Design or CASE Tool

The database design tool used was MySQL Workbench, this was a very practical and productive choice due to cost and documentation.

3.7 Other Possible E/R Relationships

When beginning to design the database, I realized there were many options for modeling the relationships between entities. The most challenging was the sale order and how to deal with the subsequent details of a single order. The solution I determined to be more effective was to have an entirely separate table for the details of a sales order and model a relation to the sale order table which houses higher level attributes of the sales order.

I also briefly considered using NoSQL to model my bike store database but I realized that relational database didn't impact performance for this use case. Additionally, the structure of this database more closely resembled a relational database as opposed to a database which didn't require relations.

4. Implementation Description

4.1 Data Dictionary

Company_Project:

Field	Type	Null	Key	Default	Extra
Project_ID	int	NO	PRI	NULL	
Project_Name	varchar(25)	NO		NULL	

Customer:

Field	Type	Null	Key	Default	Extra
Customer_ID	int	NO	PRI	NULL	
First_Name	varchar(25)	NO		NULL	
Last_Name	varchar(25)	NO		NULL	
Address	varchar(25)	NO		NULL	
City	varchar(25)	NO		NULL	
State	char(2)	NO		NULL	
Zip	varchar(5)	NO		NULL	
Current_Balance	float(7,2)	NO		NULL	

Customer_Payment:

Field	Type	Null	Key	Default	Extra
Customer_Payment_ID	int	NO	PRI	NULL	
Customer_ID	int	NO	MUL	NULL	
Payment_Type_ID	int	NO	MUL	NULL	

Department:

Field	Type	Null	Key	Default	Extra
Department_ID	int	NO	PRI	NULL	
Department_Name	varchar(25)	NO		NULL	

Dependent:

Field	Type	Null	Key	Default	Extra
Dependent_ID	int	NO	PRI	NULL	
Employee_ID	int	NO	MUL	NULL	
First_Name	varchar(25)	NO		NULL	

Last_Name	varchar(25)	NO		NULL	
Employee_Dependent_Relation_ID	int	NO	MUL	NULL	

Employee:

Field	Type	Null	Key	Default	Extra
Employee_ID	int	NO	PRI	NULL	
First_Name	varchar(25)	NO		NULL	
Last_Name	varchar(25)	NO		NULL	
Job_Title_ID	int	NO	MUL	NULL	
DOB	date	NO		NULL	
Hire_Date	date	NO		NULL	
Primary_Phone	varchar(10)	NO		NULL	
Supervisor_Employee_ID	int	YES	MUL	NULL	
Department_ID	int	NO	MUL	NULL	

Employee_Company_Project

Field	Type	Null	Key	Default	Extra
Employee_ID	int	NO	PRI	NULL	
Project_ID	int	NO	PRI	NULL	

Employee_Dependent_Relation

Field	Type	Null	Key	Default	Extra
Employee_Dependent_Relation_ID	int	NO	PRI	NULL	
Employee_Dependent_Relation_Name	varchar(25)	NO		NULL	

Job_Title

Field	Type	Null	Key	Default	Extra
Job_Title_ID	int	NO	PRI	NULL	
Job_Title_Name	varchar(25)	NO		NULL	

Payment_Type:

Field	Type	Null	Key	Default	Extra
Payment_Type_ID	int	NO	PRI	NULL	
Payment_Type_Name	varchar(25)	NO		NULL	

Product:

Field	Type	Null	Key	Default	Extra
Product_ID	int	NO	PRI	NULL	
Category_ID	int	NO	MUL	NULL	
Product_Name	varchar(25)	NO		NULL	
Product_Price	float(7,2)	NO		NULL	

Units_In_Stock	int	NO		NULL	

Product_Category

Field	Type	Null	Key	Default	Extra
Category_ID	int	NO	PRI	NULL	
Category_Name	varchar(25)	NO		NULL	

Product_Wholesaler

Field	Type	Null	Key	Default	Extra
Product_ID	int	NO	PRI	NULL	
Wholesaler_ID	int	NO	PRI	NULL	
WSP	float(7,2)	NO		NULL	

Sales_Order:

Field	Type	Null	Key	Default	Extra
Sales_Order_ID	int	NO	PRI	NULL	
Customer_ID	int	NO	MUL	NULL	
Sales_Order_Date	date	NO		NULL	
Customer_Payment_ID	int	NO	MUL	NULL	
Employee_ID	int	NO	MUL	NULL	

Sales_Order_Detail:

Field	Type	Null	Key	Default	Extra
Sales_Order_Detail_ID	int	NO	PRI	NULL	
Sales_Order_ID	int	NO	MUL	NULL	
Product_ID	int	NO	MUL	NULL	
Price	float(7,2)	NO		NULL	
Units	int	NO		NULL	

Wholesaler:

Field	Type	Null	Key	Default	Extra
Wholesaler_ID	int	NO	PRI	NULL	
Wholesaler_Name	varchar(25)	NO		NULL	
Contact	varchar(25)	NO		NULL	
Phone	varchar(10)	NO		NULL	
Email	varchar(25)	NO		NULL	

4.2 Advanced Features

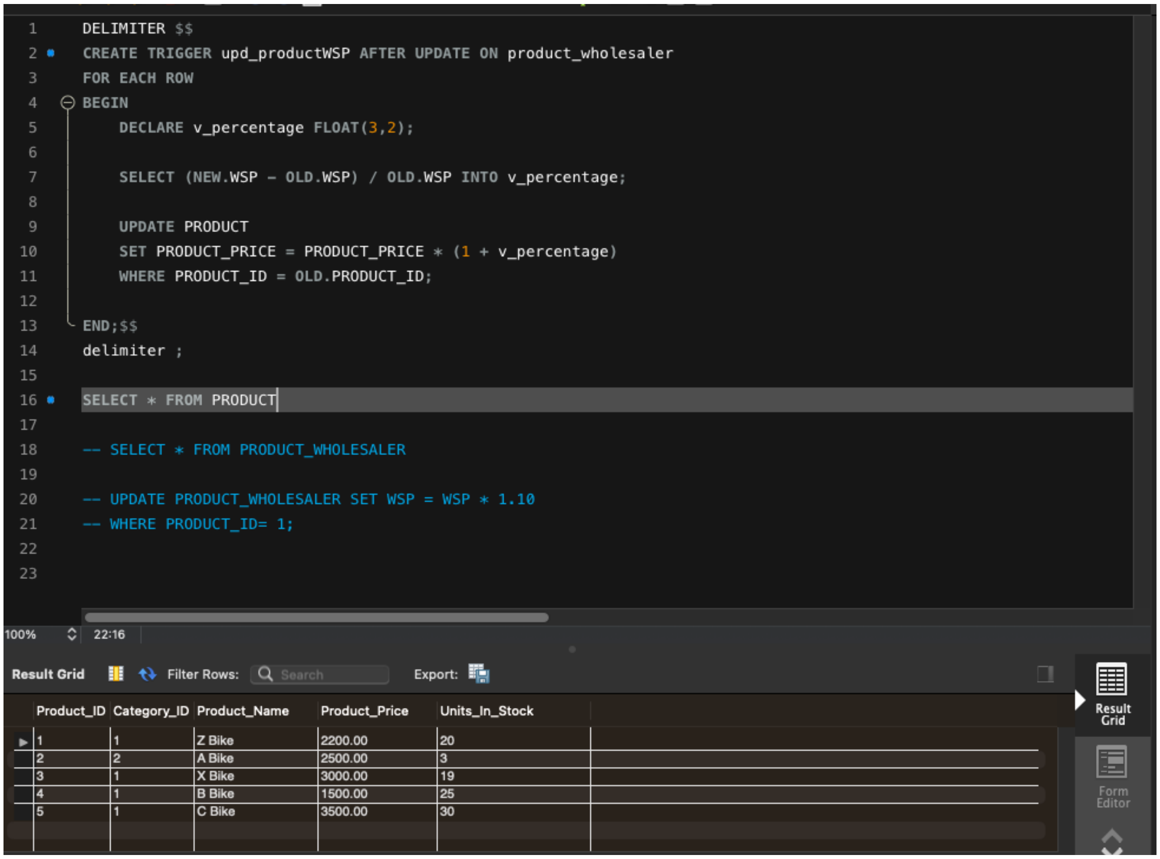
Triggers:

Advance trigger (ins_salesOrderDetail): Insert Trigger in Sales_Order_Detail – Will automatically decrement the Unit_In_Stock in product table. Furthermore it will prevent a sale if the sales order unit is greater than the product Unit_In_Stock

```
DELIMITER $$
CREATE TRIGGER ins_salesOrderDetail BEFORE INSERT ON sales_order_detail
FOR EACH ROW
BEGIN
        DECLARE v_unitsInStock INT;

        SELECT units_in_stock INTO v_unitsInStock
        FROM product
        WHERE product_id = NEW.product_id;

        IF v_unitsInStock < NEW.units THEN
                SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Not enough stock.';
        ELSE
                UPDATE product
        SET units_in_stock = units_in_stock - NEW.units;
                END IF;

END;$$
delimiter ;
```

Advance trigger (upd_productWSP) : Insert Trigger in product_wholesaler – This procedure will automatically update the product_wholesaler table by the same percentage as an update made by the wholesalers update to the product they are supplying the bikestore.

```
DELIMITER $$
CREATE TRIGGER upd_productWSP AFTER UPDATE ON product_wholesaler
FOR EACH ROW
BEGIN
        DECLARE v_percentage FLOAT(3,2);

        SELECT (NEW.WSP - OLD.WSP) / OLD.WSP INTO v_percentage;

        UPDATE PRODUCT
        SET PRODUCT_PRICE = PRODUCT_PRICE * (1 + v_percentage)
        WHERE PRODUCT_ID = OLD.PRODUCT_ID;

END;$$
delimiter ;
```

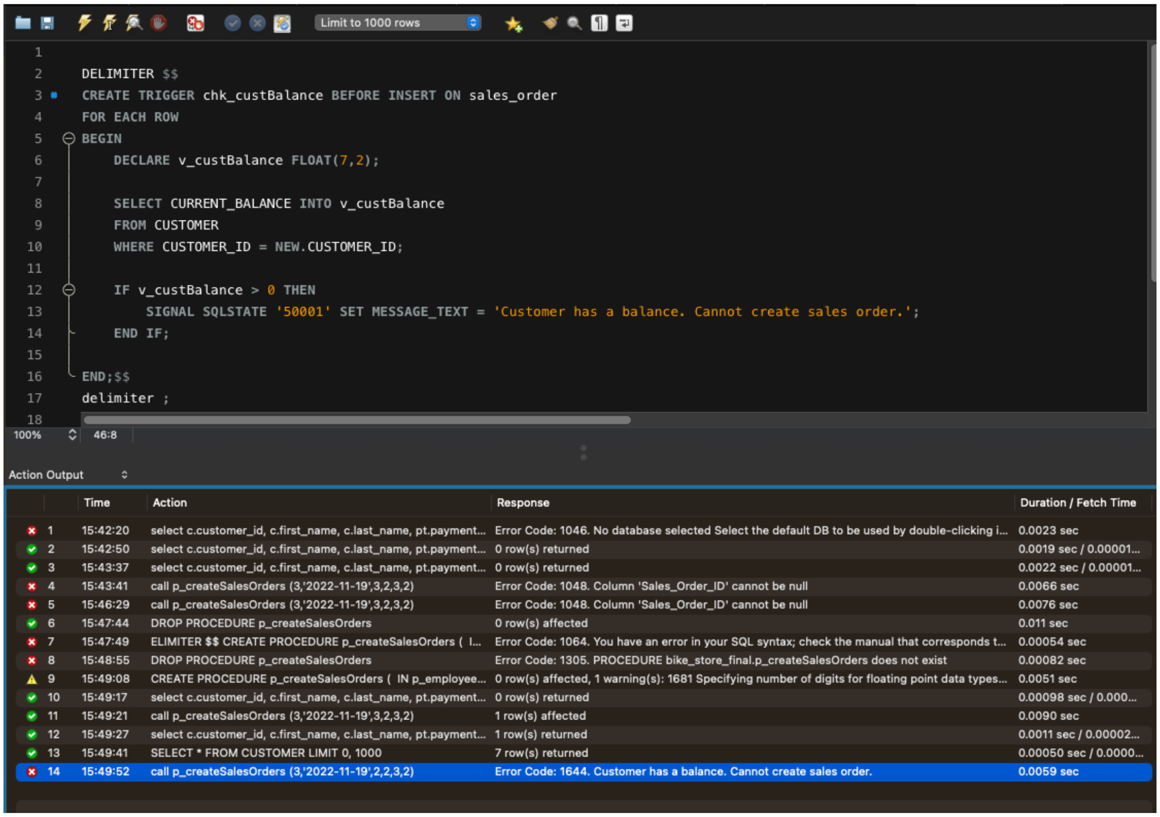


```
DELIMITER $$
CREATE TRIGGER upd_productWSP AFTER UPDATE ON product_wholesaler
FOR EACH ROW
BEGIN
    DECLARE v_percentage FLOAT(3,2);

    SELECT (NEW.WSP - OLD.WSP) / OLD.WSP INTO v_percentage;

    UPDATE PRODUCT
    SET PRODUCT_PRICE = PRODUCT_PRICE * (1 + v_percentage)
    WHERE PRODUCT_ID = OLD.PRODUCT_ID;

END;$$
delimiter ;

SELECT * FROM PRODUCT

-- SELECT * FROM PRODUCT_WHOLESALER

-- UPDATE PRODUCT_WHOLESALER SET WSP = WSP * 1.10
-- WHERE PRODUCT_ID= 1;
```

Product_ID	Category_ID	Product_Name	Product_Price	Units_In_Stock
1	1	Z Bike	2200.00	20
2	2	A Bike	2500.00	3
3	1	X Bike	3000.00	19
4	1	B Bike	1500.00	25
5	1	C Bike	3500.00	30

Advance trigger (chk_custBalance) : Process to confirm if a customer has outstanding balance – This trigger will automatically check to see if a customer attempting to purchase an additional bike already has an outstanding balance. If the customer does have an outstanding balance and error will be returned “Error Code: 1644. Customer has a balance. Cannot create sales order.”

```
DELIMITER $$
CREATE TRIGGER chk_custBalance BEFORE INSERT ON sales_order
FOR EACH ROW
BEGIN
        DECLARE v_custBalance FLOAT(7,2);

        SELECT CURRENT_BALANCE INTO v_custBalance
        FROM CUSTOMER
        WHERE CUSTOMER_ID = NEW.CUSTOMER_ID;

        IF v_custBalance > 0 THEN
                SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Customer has a balance.
        Cannot create sales order.';
        END IF;

END;$$
DELIMITER ;
```



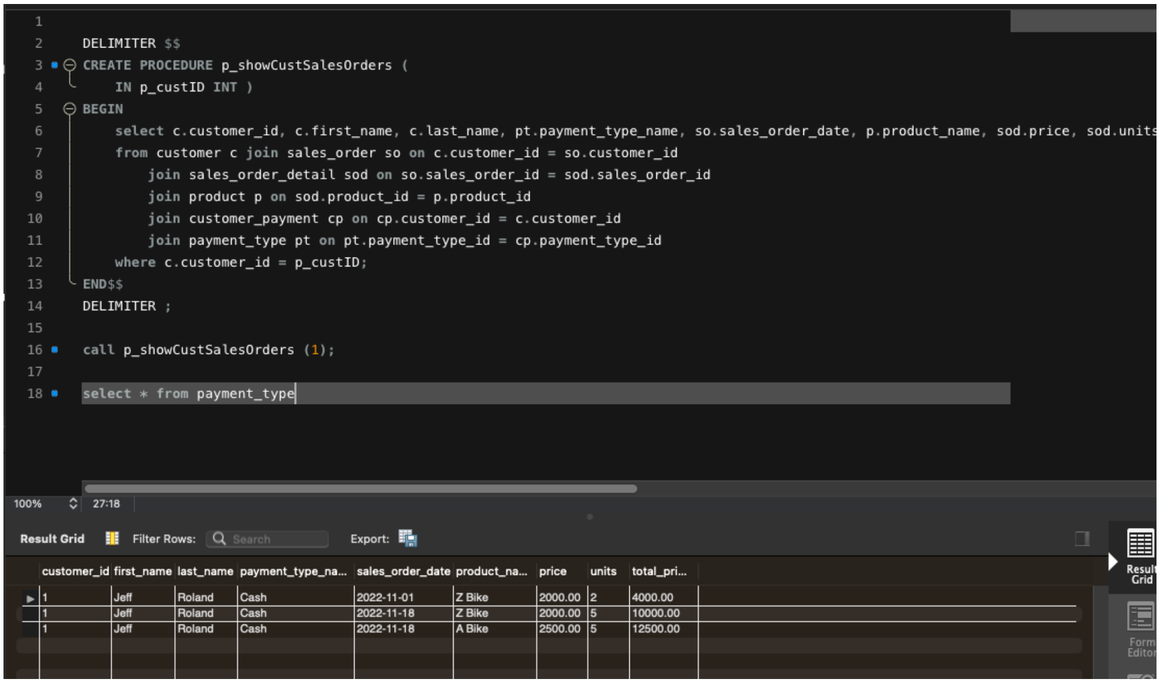
```

DELIMITER $$
CREATE TRIGGER chk_custBalance BEFORE INSERT ON sales_order
FOR EACH ROW
BEGIN
    DECLARE v_custBalance FLOAT(7,2);

    SELECT CURRENT_BALANCE INTO v_custBalance
    FROM CUSTOMER
    WHERE CUSTOMER_ID = NEW.CUSTOMER_ID;

    IF v_custBalance > 0 THEN
        SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Customer has a balance. Cannot create sales order.';
    END IF;

END;$$
delimiter ;

```

		Time	Action	Response	Duration / Fetch Time
✗	1	15:42:20	select c.customer_id, c.first_name, c.last_name, pt.payment...	Error Code: 1046. No database selected Select the default DB to be used by double-clicking i...	0.0023 sec
✓	2	15:42:50	select c.customer_id, c.first_name, c.last_name, pt.payment...	0 row(s) returned	0.0019 sec / 0.00001...
✓	3	15:43:37	select c.customer_id, c.first_name, c.last_name, pt.payment...	0 row(s) returned	0.0022 sec / 0.00001...
✗	4	15:43:41	call p_createSalesOrders (3,'2022-11-19',3,2,3,2)	Error Code: 1048. Column 'Sales_Order_ID' cannot be null	0.0066 sec
✗	5	15:46:29	call p_createSalesOrders (3,'2022-11-19',3,2,3,2)	Error Code: 1048. Column 'Sales_Order_ID' cannot be null	0.0076 sec
✓	6	15:47:44	DROP PROCEDURE p_createSalesOrders	0 row(s) affected	0.011 sec
✗	7	15:47:49	ELIMITER $$ CREATE PROCEDURE p_createSalesOrders (I...	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds t...	0.00054 sec
✗	8	15:48:55	DROP PROCEDURE p_createSalesOrders	Error Code: 1305. PROCEDURE bike_store_final.p_createSalesOrders does not exist	0.00082 sec
⚠	9	15:49:08	CREATE PROCEDURE p_createSalesOrders (IN p_employee...	0 row(s) affected, 1 warning(s): 1681 Specifying number of digits for floating point data types...	0.0051 sec
✓	10	15:49:17	select c.customer_id, c.first_name, c.last_name, pt.payment...	0 row(s) returned	0.00098 sec / 0.000...
✓	11	15:49:21	call p_createSalesOrders (3,'2022-11-19',3,2,3,2)	1 row(s) affected	0.0090 sec
✓	12	15:49:27	select c.customer_id, c.first_name, c.last_name, pt.payment...	1 row(s) returned	0.0011 sec / 0.00002...
✓	13	15:49:41	SELECT * FROM CUSTOMER LIMIT 0, 1000	7 row(s) returned	0.00050 sec / 0.0000...
✗	14	15:49:52	call p_createSalesOrders (3,'2022-11-19',2,2,3,2)	Error Code: 1644. Customer has a balance. Cannot create sales order.	0.0059 sec

Stored Procedure:

Advance stored procedure (p_showCustSalesOrders)**:** Process to show customer sales orders. This procedure will automatically display all customer sales orders by calling the procedure using a customer's ID.

```
DELIMITER $$
CREATE PROCEDURE p_showCustSalesOrders (
        IN p_custID INT )
BEGIN
        select c.customer_id, c.first_name, c.last_name, pt.payment_type_name, so.sales_order_date,
p.product_name, sod.price, sod.units, sod.price * sod.units as total_price
        from customer c join sales_order so on c.customer_id = so.customer_id
                join sales_order_detail sod on so.sales_order_id = sod.sales_order_id
                join product p on sod.product_id = p.product_id
                join customer_payment cp on cp.customer_id = c.customer_id
    join payment_type pt on pt.payment_type_id = cp.payment_type_id
        where c.customer_id = p_custID;
END$$
DELIMITER ;
```



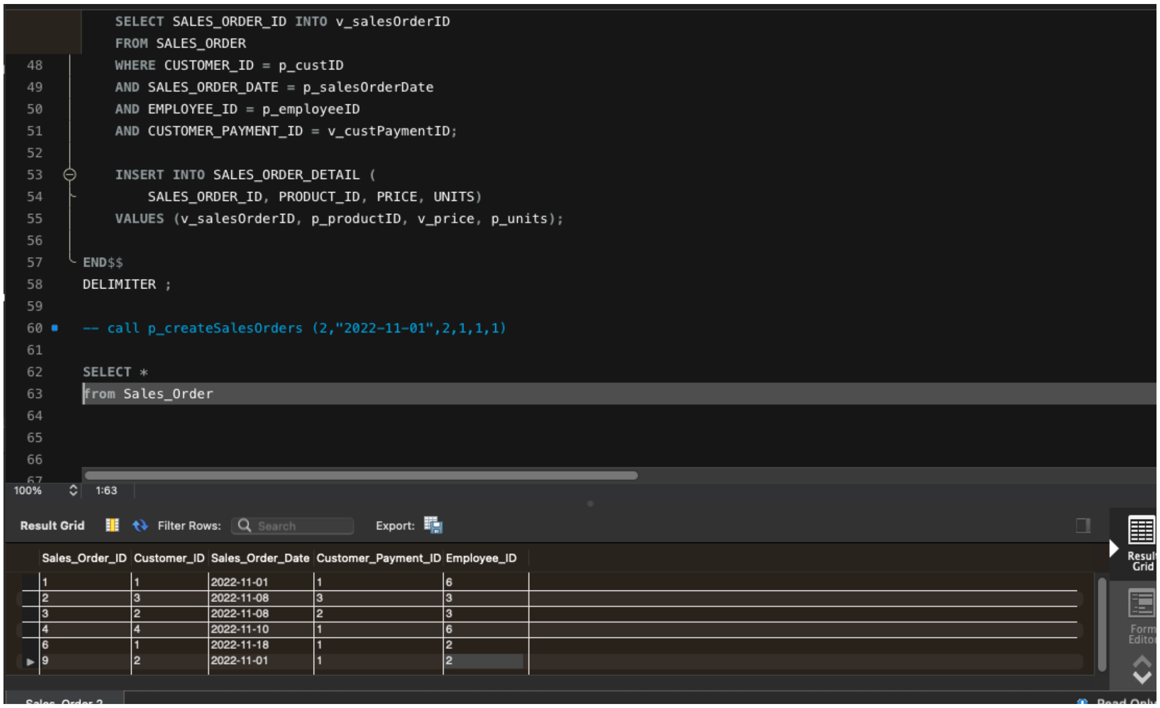
```
DELIMITER $$
CREATE PROCEDURE p_showCustSalesOrders (
    IN p_custID INT )
BEGIN
    select c.customer_id, c.first_name, c.last_name, pt.payment_type_name, so.sales_order_date, p.product_name, sod.price, sod.units
    from customer c join sales_order so on c.customer_id = so.customer_id
        join sales_order_detail sod on so.sales_order_id = sod.sales_order_id
        join product p on sod.product_id = p.product_id
        join customer_payment cp on cp.customer_id = c.customer_id
        join payment_type pt on pt.payment_type_id = cp.payment_type_id
    where c.customer_id = p_custID;
END$$
DELIMITER ;

call p_showCustSalesOrders (1);

select * from payment_type
```

customer_id	first_name	last_name	payment_type_na...	sales_order_date	product_na...	price	units	total_pri...
1	Jeff	Roland	Cash	2022-11-01	Z Bike	2000.00	2	4000.00
1	Jeff	Roland	Cash	2022-11-18	Z Bike	2000.00	5	10000.00
1	Jeff	Roland	Cash	2022-11-18	A Bike	2500.00	5	12500.00

Advance stored procedure (p_createSalesOrders): Process to create a sales order.
This process will generate a sales order and sales order details based on the parameters provided. Process will first check to see if sales order already exists for same customer/order-date/employee/payment. If so, it will link the sales-order-detail to that sales-order.

```
DELIMITER $$

CREATE PROCEDURE p_createSalesOrders (
        IN p_employeeID INT,
        IN p_salesOrderDate DATE,
        IN p_custID INT,
        IN p_paymentTypeID INT,
        IN p_productID INT,
        IN p_units INT )
BEGIN

        DECLARE v_salesOrderID INT;
        DECLARE v_custPaymentID INT;
        DECLARE v_price FLOAT(7,2);

        SELECT CUSTOMER_PAYMENT_ID INTO v_custPaymentID
        FROM CUSTOMER_PAYMENT
        WHERE CUSTOMER_ID = p_custID
        AND PAYMENT_TYPE_ID = p_paymentTypeID;

        IF v_custPaymentID IS NULL THEN

                INSERT INTO CUSTOMER_PAYMENT (CUSTOMER_ID, PAYMENT_TYPE_ID)
                VALUES (p_custID, p_paymentTypeID);

        END IF;

        SELECT SALES_ORDER_ID INTO v_salesOrderID
```

```
    FROM SALES_ORDER
    WHERE CUSTOMER_ID = p_custID
    AND SALES_ORDER_DATE = p_salesOrderDate
    AND EMPLOYEE_ID = p_employeeID
    AND CUSTOMER_PAYMENT_ID = v_custPaymentID;

    IF v_salesOrderID IS NULL THEN

                    SELECT MAX(SALES_ORDER_ID) + 1 INTO v_salesOrderID
                    FROM SALES_ORDER;

                    INSERT INTO SALES_ORDER (
                            SALES_ORDER_ID, CUSTOMER_ID, SALES_ORDER_DATE,
                    CUSTOMER_PAYMENT_ID, EMPLOYEE_ID)
                    VALUES (v_salesOrderID, p_custID, p_salesOrderDate, p_paymentTypeID,
                    p_employeeID);

                    END IF;

    SELECT PRODUCT_PRICE INTO v_price
    FROM PRODUCT
    WHERE PRODUCT_ID = p_productID;

    INSERT INTO SALES_ORDER_DETAIL (
                    SALES_ORDER_ID, PRODUCT_ID, PRICE, UNITS)
          VALUES (v_salesOrderID, p_productID, v_price, p_units);

END$$
DELIMITER ;
```

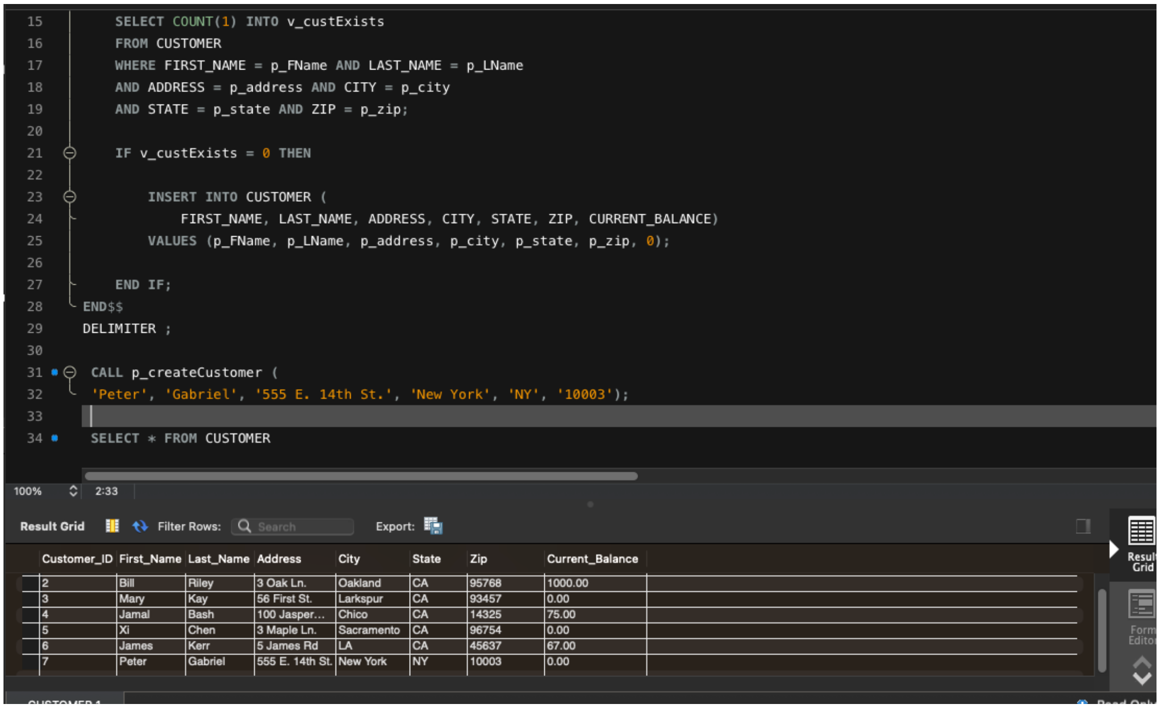


```
    SELECT SALES_ORDER_ID INTO v_salesOrderID
    FROM SALES_ORDER
    WHERE CUSTOMER_ID = p_custID
    AND SALES_ORDER_DATE = p_salesOrderDate
    AND EMPLOYEE_ID = p_employeeID
    AND CUSTOMER_PAYMENT_ID = v_custPaymentID;

    INSERT INTO SALES_ORDER_DETAIL (
        SALES_ORDER_ID, PRODUCT_ID, PRICE, UNITS)
    VALUES (v_salesOrderID, p_productID, v_price, p_units);

END$$
DELIMITER ;

-- call p_createSalesOrders (2,"2022-11-01",2,1,1,1)

SELECT *
from Sales_Order
```

Sales_Order_ID	Customer_ID	Sales_Order_Date	Customer_Payment_ID	Employee_ID
1	1	2022-11-01	1	6
2	3	2022-11-08	3	3
3	2	2022-11-08	2	3
4	4	2022-11-10	1	6
6	1	2022-11-18	1	2
9	2	2022-11-01	1	2

Advance stored procedure (p_createCustomer)**:** This process will add a new customer. It will first check to see if the customer already exists and if so, will not insert.

```
DELIMITER $$

CREATE PROCEDURE p_createCustomer (
        IN p_FName VARCHAR(25),
        IN p_LName VARCHAR(25),
        IN p_address VARCHAR(25),
        IN p_city VARCHAR(25),
        IN p_state CHAR(2),
        IN p_zip VARCHAR(5) )
BEGIN
        DECLARE v_custExists INT;

        SELECT COUNT(1) INTO v_custExists
        FROM CUSTOMER
        WHERE FIRST_NAME = p_FName AND LAST_NAME = p_LName
        AND ADDRESS = p_address AND CITY = p_city
        AND STATE = p_state AND ZIP = p_zip;

        IF v_custExists = 0 THEN

                INSERT INTO CUSTOMER (
                        FIRST_NAME, LAST_NAME, ADDRESS, CITY, STATE, ZIP,
                        CURRENT_BALANCE)
                VALUES (p_FName, p_LName, p_address, p_city, p_state, p_zip, 0);

        END IF;
END$$
DELIMITER ;
```



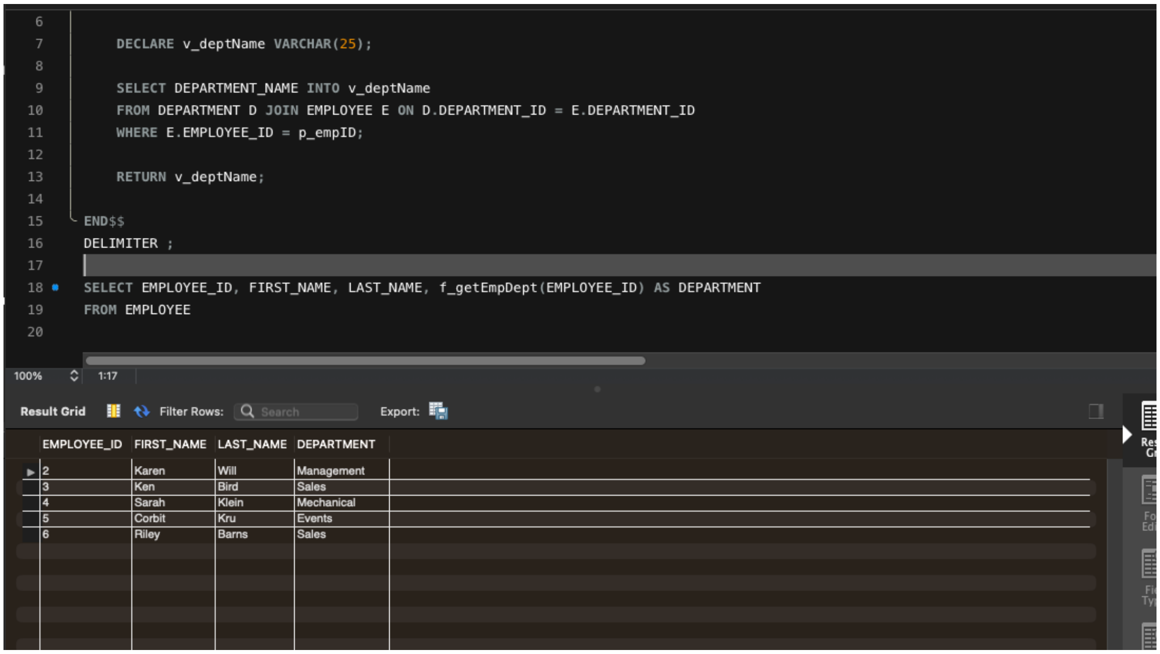
```
    SELECT COUNT(1) INTO v_custExists
    FROM CUSTOMER
    WHERE FIRST_NAME = p_FName AND LAST_NAME = p_LName
    AND ADDRESS = p_address AND CITY = p_city
    AND STATE = p_state AND ZIP = p_zip;

    IF v_custExists = 0 THEN

        INSERT INTO CUSTOMER (
            FIRST_NAME, LAST_NAME, ADDRESS, CITY, STATE, ZIP, CURRENT_BALANCE)
        VALUES (p_FName, p_LName, p_address, p_city, p_state, p_zip, 0);

    END IF;
END$$
DELIMITER ;

CALL p_createCustomer (
'Peter', 'Gabriel', '555 E. 14th St.', 'New York', 'NY', '10003');

SELECT * FROM CUSTOMER
```

Customer_ID	First_Name	Last_Name	Address	City	State	Zip	Current_Balance
2	Bill	Riley	3 Oak Ln.	Oakland	CA	95768	1000.00
3	Mary	Kay	56 First St.	Larkspur	CA	93457	0.00
4	Jamal	Bash	100 Jasper...	Chico	CA	14325	75.00
5	Xi	Chen	3 Maple Ln.	Sacramento	CA	96754	0.00
6	James	Kerr	5 James Rd	LA	CA	45637	67.00
7	Peter	Gabriel	555 E. 14th St.	New York	NY	10003	0.00

Functions:

Advance function (f_getEmpDept): This function returns the department associated with each employee ID.

```
DELIMITER $$
CREATE FUNCTION f_getEmpDept ( p_EmpID INT ) RETURNS VARCHAR(25) DETERMINISTIC

BEGIN

        DECLARE v_deptName VARCHAR(25);

        SELECT DEPARTMENT_NAME INTO v_deptName
        FROM DEPARTMENT D JOIN EMPLOYEE E ON D.DEPARTMENT_ID =
        E.DEPARTMENT_ID
        WHERE E.EMPLOYEE_ID = p_empID;

        RETURN v_deptName;

END$$
DELIMITER ;
```

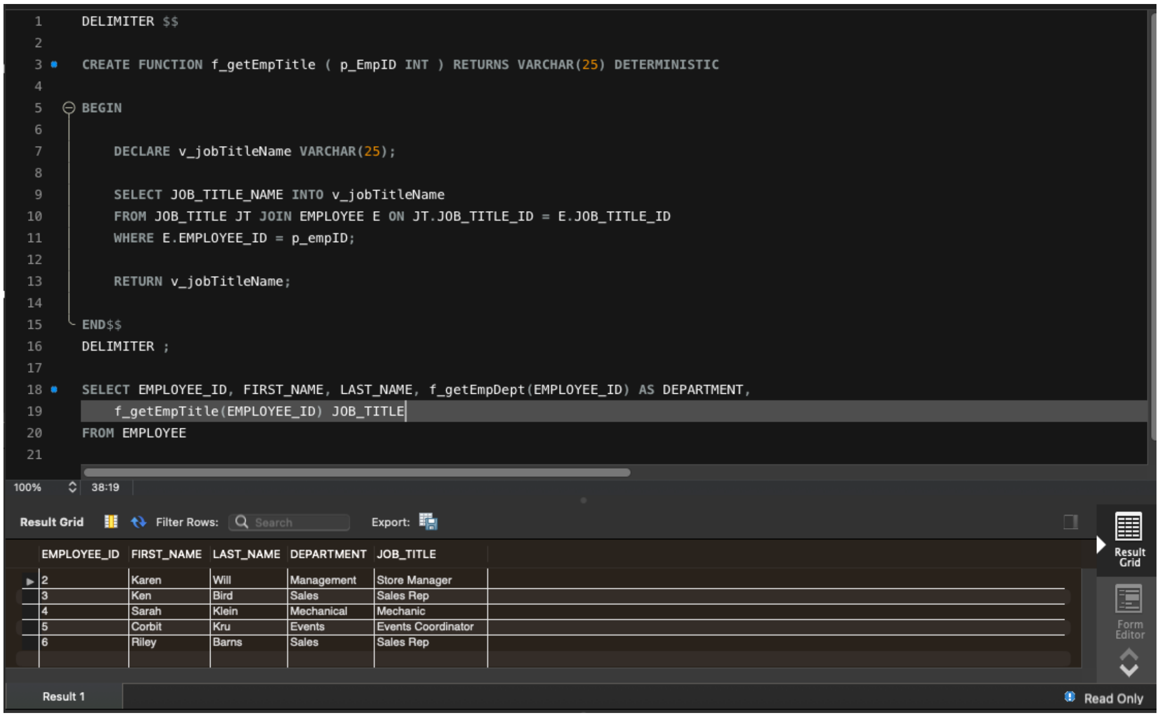


```

    DECLARE v_deptName VARCHAR(25);

    SELECT DEPARTMENT_NAME INTO v_deptName
    FROM DEPARTMENT D JOIN EMPLOYEE E ON D.DEPARTMENT_ID = E.DEPARTMENT_ID
    WHERE E.EMPLOYEE_ID = p_empID;

    RETURN v_deptName;

END$$
DELIMITER ;

SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, f_getEmpDept(EMPLOYEE_ID) AS DEPARTMENT
FROM EMPLOYEE
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT
2	Karen	Will	Management
3	Ken	Bird	Sales
4	Sarah	Klein	Mechanical
5	Corbit	Kru	Events
6	Riley	Barns	Sales

Advance function (f_getEmpTitle): This function returns the employee's title associated with each employee ID.

```
DELIMITER $$

CREATE FUNCTION f_getEmpTitle ( p_EmpID INT ) RETURNS VARCHAR(25) DETERMINISTIC

BEGIN

        DECLARE v_jobTitleName VARCHAR(25);
        SELECT JOB_TITLE_NAME INTO v_jobTitleName
        FROM JOB_TITLE JT JOIN EMPLOYEE E ON JT.JOB_TITLE_ID = E.JOB_TITLE_ID
        WHERE E.EMPLOYEE_ID = p_empID;

        RETURN v_jobTitleName;

END$$
DELIMITER ;
```

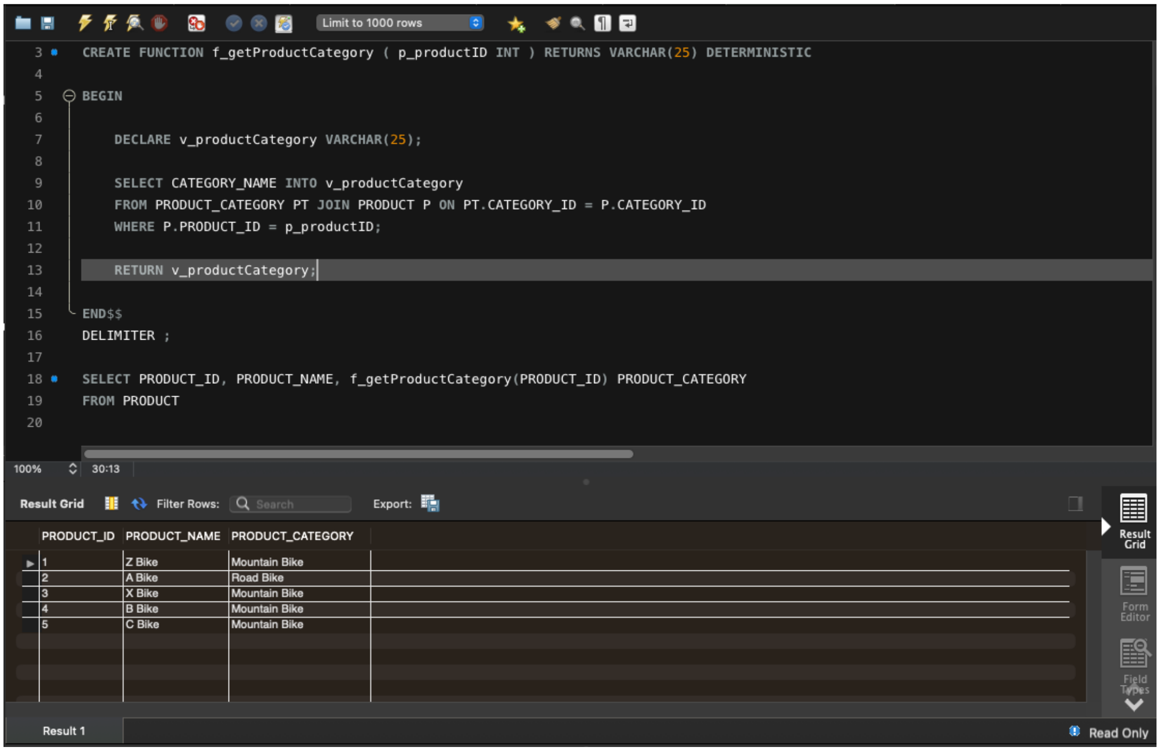


```
DELIMITER $$

CREATE FUNCTION f_getEmpTitle ( p_EmpID INT ) RETURNS VARCHAR(25) DETERMINISTIC

BEGIN

    DECLARE v_jobTitleName VARCHAR(25);

    SELECT JOB_TITLE_NAME INTO v_jobTitleName
    FROM JOB_TITLE JT JOIN EMPLOYEE E ON JT.JOB_TITLE_ID = E.JOB_TITLE_ID
    WHERE E.EMPLOYEE_ID = p_empID;

    RETURN v_jobTitleName;

END$$
DELIMITER ;

SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, f_getEmpDept(EMPLOYEE_ID) AS DEPARTMENT,
    f_getEmpTitle(EMPLOYEE_ID) JOB_TITLE
FROM EMPLOYEE
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT	JOB_TITLE
2	Karen	Will	Management	Store Manager
3	Ken	Bird	Sales	Sales Rep
4	Sarah	Klein	Mechanical	Mechanic
5	Corbit	Kru	Events	Events Coordinator
6	Riley	Barns	Sales	Sales Rep

Advance function (f_getProductCategory): This function returns the product category associated with each product ID.

```
DELIMITER $$

CREATE FUNCTION f_getProductCategory ( p_productID INT ) RETURNS VARCHAR(25)
DETERMINISTIC

BEGIN

        DECLARE v_productCategory VARCHAR(25);

        SELECT CATEGORY_NAME INTO v_productCategory
        FROM PRODUCT_CATEGORY PT JOIN PRODUCT P ON PT.CATEGORY_ID =
        P.CATEGORY_ID
        WHERE P.PRODUCT_ID = p_productID;

        RETURN v_productCategory;

END$$
DELIMITER ;
```



```
CREATE FUNCTION f_getProductCategory ( p_productID INT ) RETURNS VARCHAR(25) DETERMINISTIC

BEGIN

    DECLARE v_productCategory VARCHAR(25);

    SELECT CATEGORY_NAME INTO v_productCategory
    FROM PRODUCT_CATEGORY PT JOIN PRODUCT P ON PT.CATEGORY_ID = P.CATEGORY_ID
    WHERE P.PRODUCT_ID = p_productID;

    RETURN v_productCategory;

END$$
DELIMITER ;

SELECT PRODUCT_ID, PRODUCT_NAME, f_getProductCategory(PRODUCT_ID) PRODUCT_CATEGORY
FROM PRODUCT
```

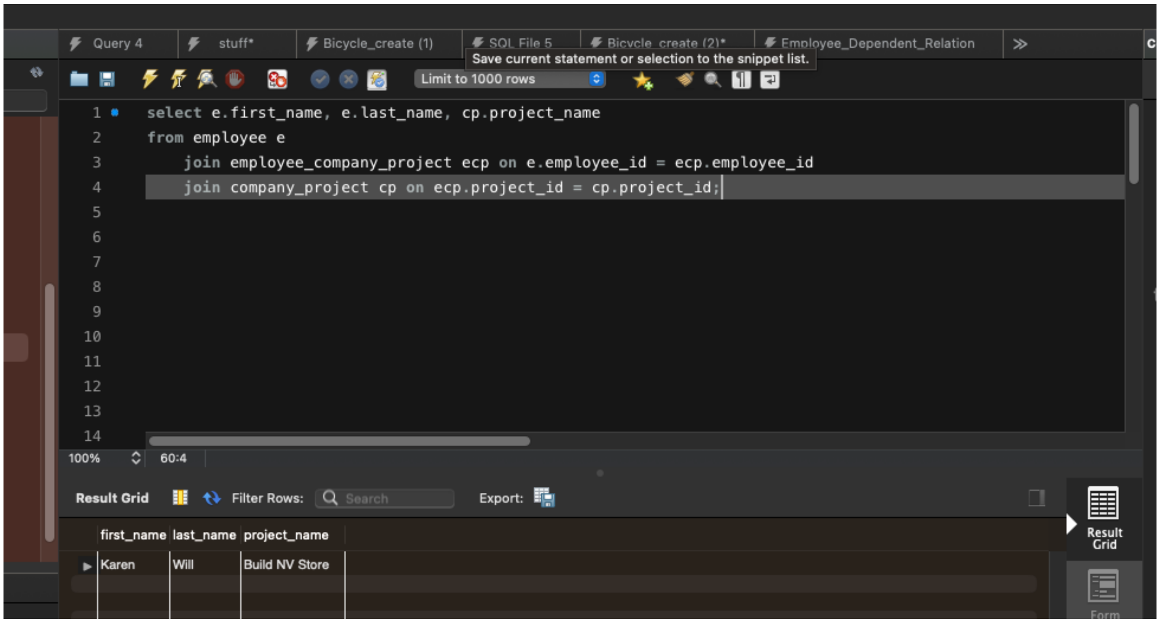
PRODUCT_ID	PRODUCT_NAME	PRODUCT_CATEGORY
1	Z Bike	Mountain Bike
2	A Bike	Road Bike
3	X Bike	Mountain Bike
4	B Bike	Mountain Bike
5	C Bike	Mountain Bike

4.3 Queries

4.3.1 List company projects and who worked on the projects

- This query will project the first and last name of each employee assigned to all projects

```
select e.first_name, e.last_name, cp.project_name
from employee e
            join employee_company_project ecp on e.employee_id = ecp.employee_id
            join company_project cp on ecp.project_id = cp.project_id;
```



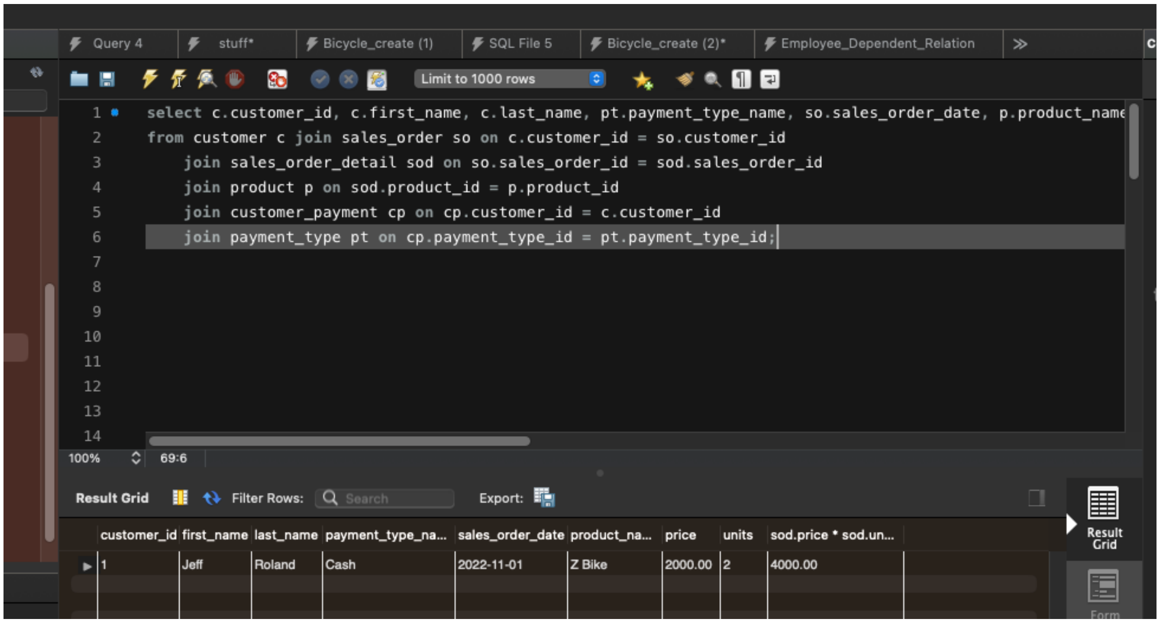
```
select e.first_name, e.last_name, cp.project_name
from employee e
    join employee_company_project ecp on e.employee_id = ecp.employee_id
    join company_project cp on ecp.project_id = cp.project_id;
```

```
first_name last_name project_name
Karen      Will      Build NV Store
```

4.3.2 Customer Purchase History

- This query will project the customer id, first name, last name, payment type used, sales date, product purchased, product price, units sold, and total value of the purchase for all customers who purchased bikes

```
select c.customer_id, c.first_name, c.last_name, pt.payment_type_name, so.sales_order_date,
p.product_name, sod.price, sod.units, sod.price * sod.units
from customer c join sales_order so on c.customer_id = so.customer_id
   join sales_order_detail sod on so.sales_order_id = sod.sales_order_id
   join product p on sod.product_id = p.product_id
   join customer_payment cp on cp.customer_id = c.customer_id
   join payment_type pt on cp.payment_type_id = pt.payment_type_id;
```



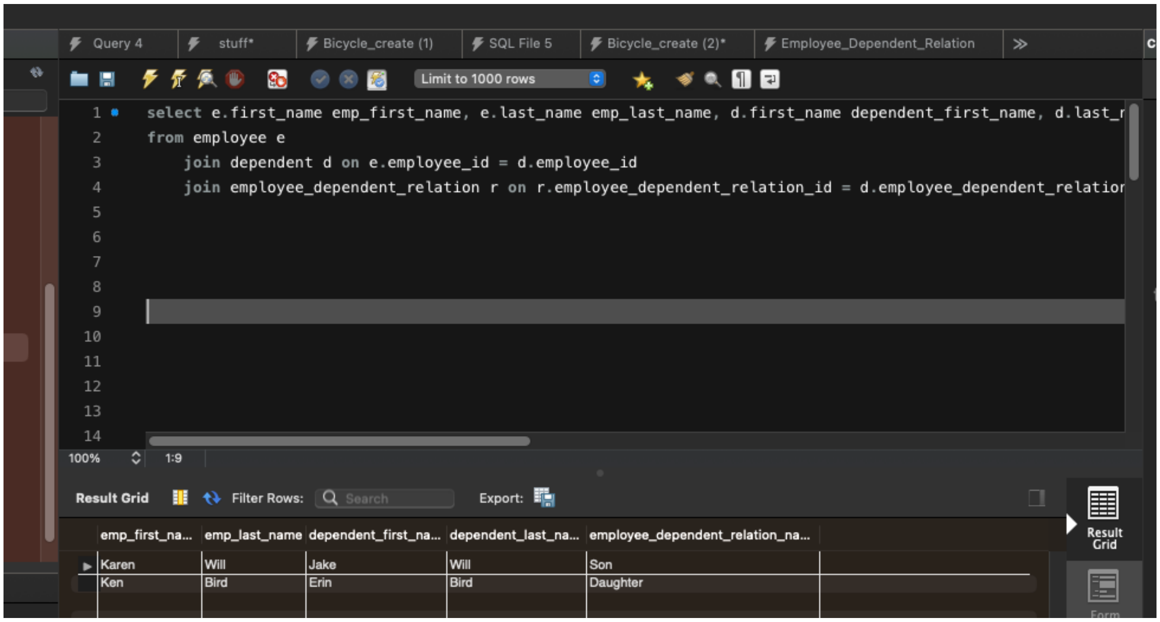
```sql
select c.customer_id, c.first_name, c.last_name, pt.payment_type_name, so.sales_order_date, p.product_name
from customer c join sales_order so on c.customer_id = so.customer_id
    join sales_order_detail sod on so.sales_order_id = sod.sales_order_id
    join product p on sod.product_id = p.product_id
    join customer_payment cp on cp.customer_id = c.customer_id
    join payment_type pt on cp.payment_type_id = pt.payment_type_id;
```

customer_id	first_name	last_name	payment_type_na...	sales_order_date	product_na...	price	units	sod.price * sod.un...
1	Jeff	Roland	Cash	2022-11-01	Z Bike	2000.00	2	4000.00

4.3.3 List employees with their dependents

- This query will project the employee first name, employee last name, dependent first name, dependent last name, as well as the relation name of each employee with dependents

```
select e.first_name emp_first_name, e.last_name emp_last_name, d.first_name dependent_first_name,
d.last_name dependent_last_name, r.employee_dependent_relation_name
from employee e
        join dependent d on e.employee_id = d.employee_id
        join employee_dependent_relation r on r.employee_dependent_relation_id =
        d.employee_dependent_relation_id;
```



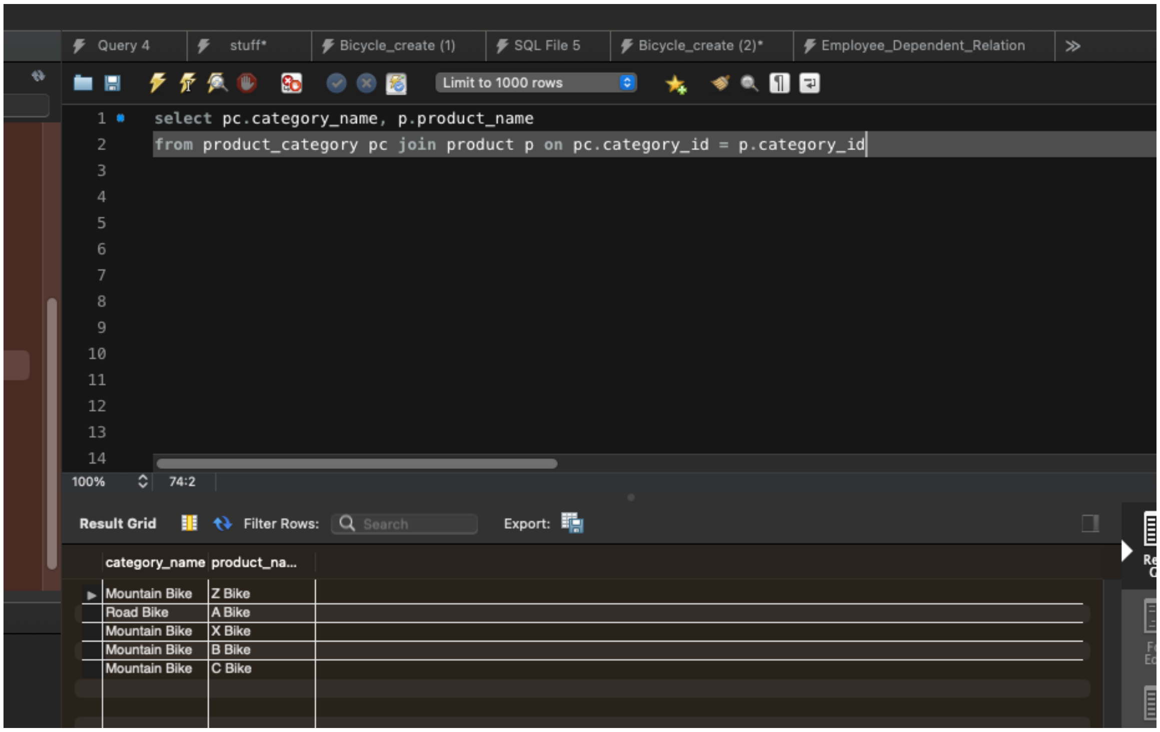
```sql
select e.first_name emp_first_name, e.last_name emp_last_name, d.first_name dependent_first_name, d.last_r
from employee e
    join dependent d on e.employee_id = d.employee_id
    join employee_dependent_relation r on r.employee_dependent_relation_id = d.employee_dependent_relatior
```

emp_first_na...	emp_last_name	dependent_first_na...	dependent_last_na...	employee_dependent_relation_na...
Karen	Will	Jake	Will	Son
Ken	Bird	Erin	Bird	Daughter

4.3.4 List all bicycles by category

- This query will project the category name and all bikes currently available in the store’s inventory

```
select pc.category_name, p.product_name
from product_category pc join product p on pc.category_id = p.category_id
```



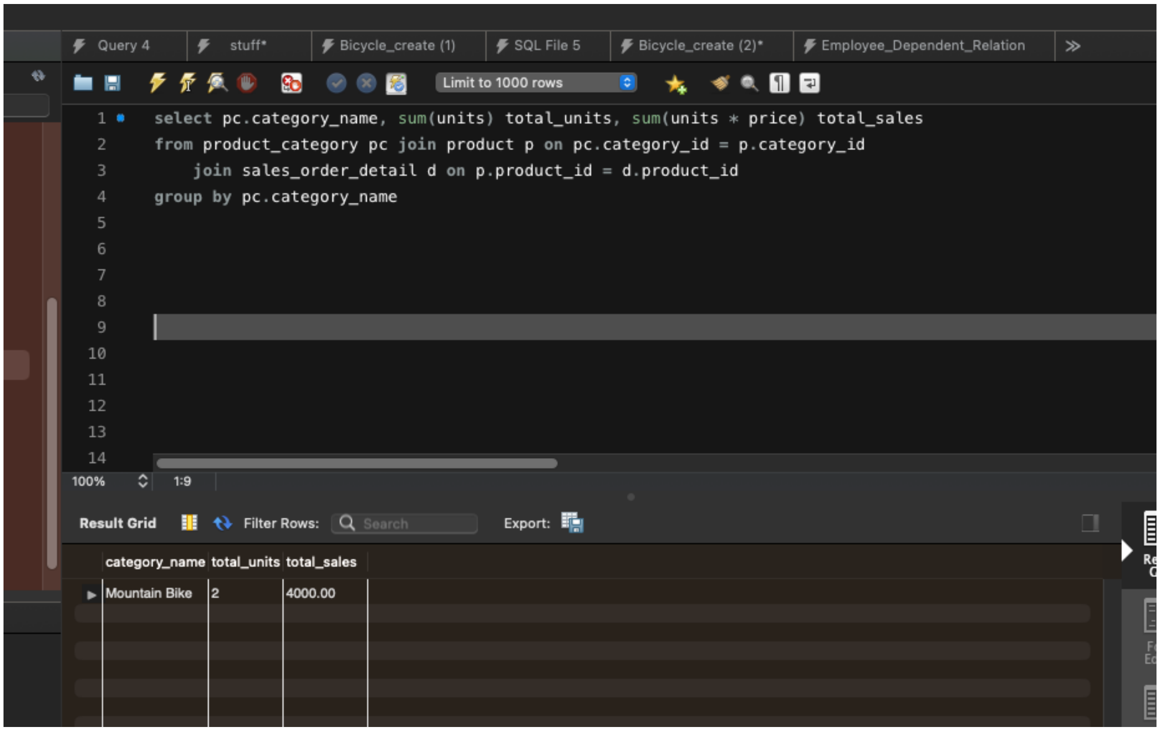
```
select pc.category_name, p.product_name
from product_category pc join product p on pc.category_id = p.category_id
```

category_name	product_na...
Mountain Bike	Z Bike
Road Bike	A Bike
Mountain Bike	X Bike
Mountain Bike	B Bike
Mountain Bike	C Bike

4.3.5 List total sales and total units sold by bicycle category

- This query will project the category name and sum up the total number of units and total market value of all bikes which have been sold for each category

```
select pc.category_name, sum(units) total_units, sum(units * price) total_sales
from product_category pc join product p on pc.category_id = p.category_id
        join sales_order_detail d on p.product_id = d.product_id
group by pc.category_name
```



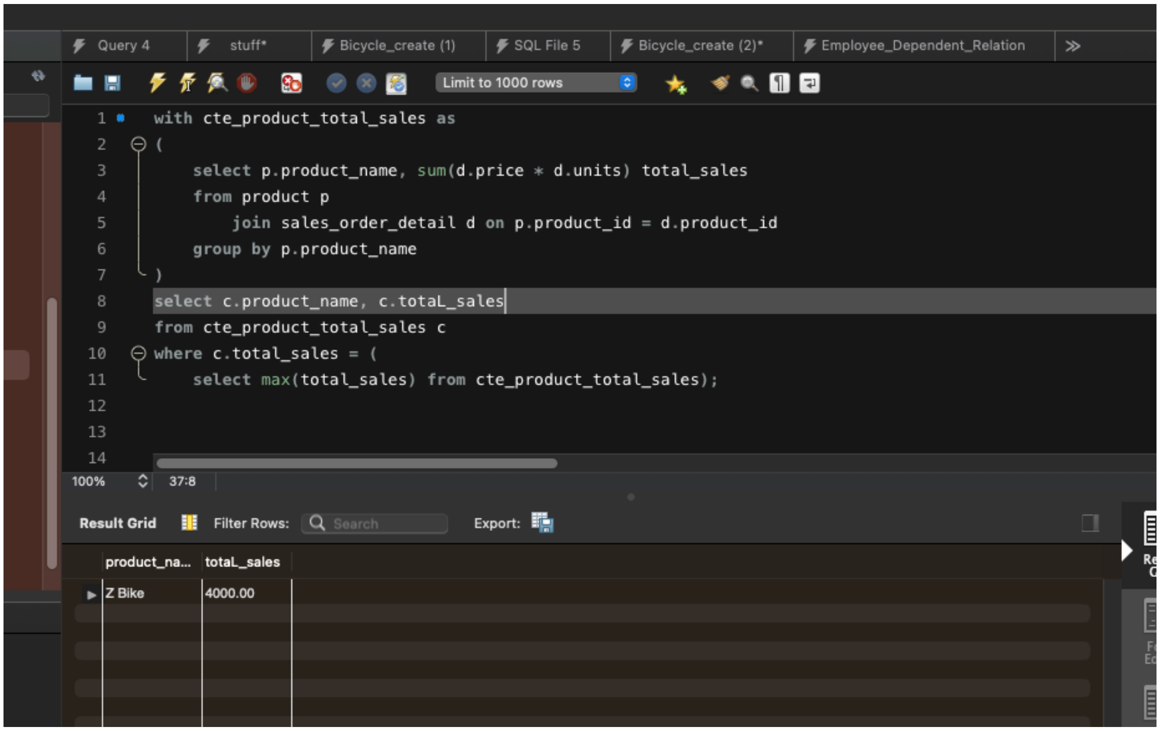
```
select pc.category_name, sum(units) total_units, sum(units * price) total_sales
from product_category pc join product p on pc.category_id = p.category_id
    join sales_order_detail d on p.product_id = d.product_id
group by pc.category_name
```

category_name	total_units	total_sales
Mountain Bike	2	4000.00

4.3.6 List best selling bicycle by revenue

- This query will project the product name of the best selling bike in the store by revenue total

```
with cte_product_total_sales as
(
        select p.product_name, sum(d.price * d.units) total_sales
        from product p
                join sales_order_detail d on p.product_id = d.product_id
        group by p.product_name
)
select c.product_name, c.totaL_sales
from cte_product_total_sales c
where c.total_sales = (
        select max(total_sales) from cte_product_total_sales);
```



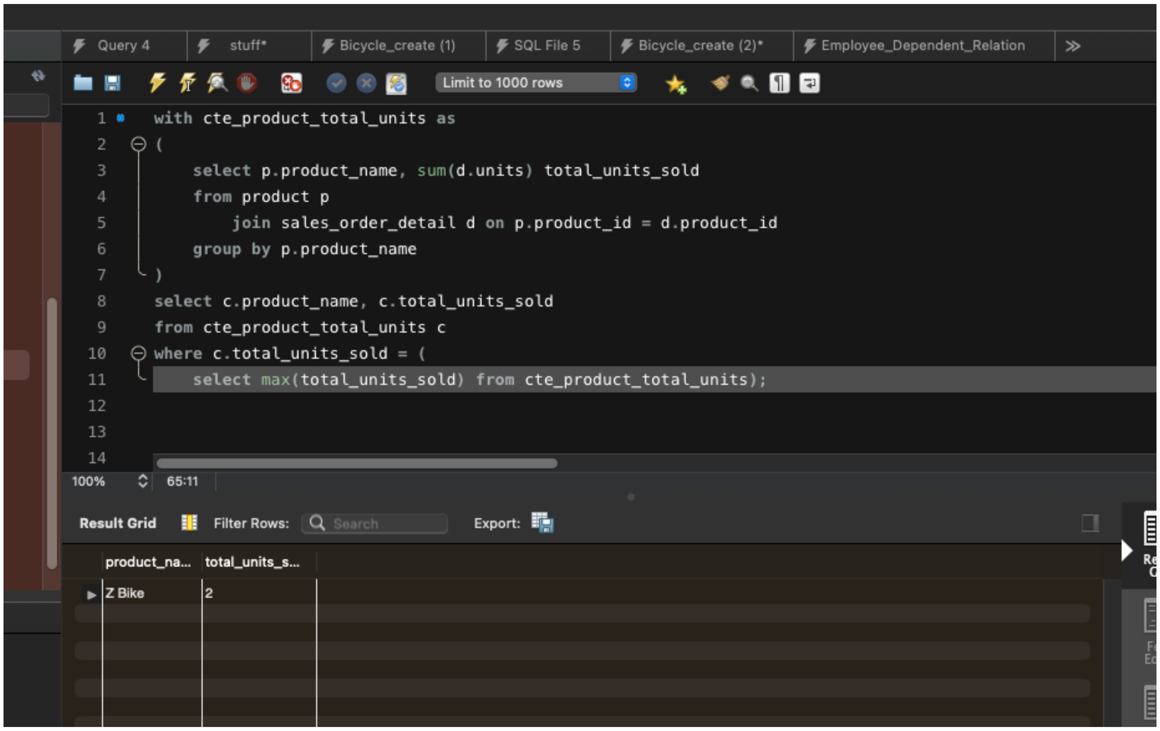
```
with cte_product_total_sales as
(
    select p.product_name, sum(d.price * d.units) total_sales
    from product p
        join sales_order_detail d on p.product_id = d.product_id
    group by p.product_name
)
select c.product_name, c.totaL_sales
from cte_product_total_sales c
where c.total_sales = (
    select max(total_sales) from cte_product_total_sales);
```

product_na...	totaL_sales
Z Bike	4000.00

4.3.7 List best selling bikes by units sold

- This query will project the best selling product name by total units sold

```
with cte_product_total_units as
(
	select p.product_name, sum(d.units) total_units_sold
	from product p
		join sales_order_detail d on p.product_id = d.product_id
	group by p.product_name
)
select c.product_name, c.total_units_sold
from cte_product_total_units c
where c.total_units_sold = (
	select max(total_units_sold) from cte_product_total_units);
```



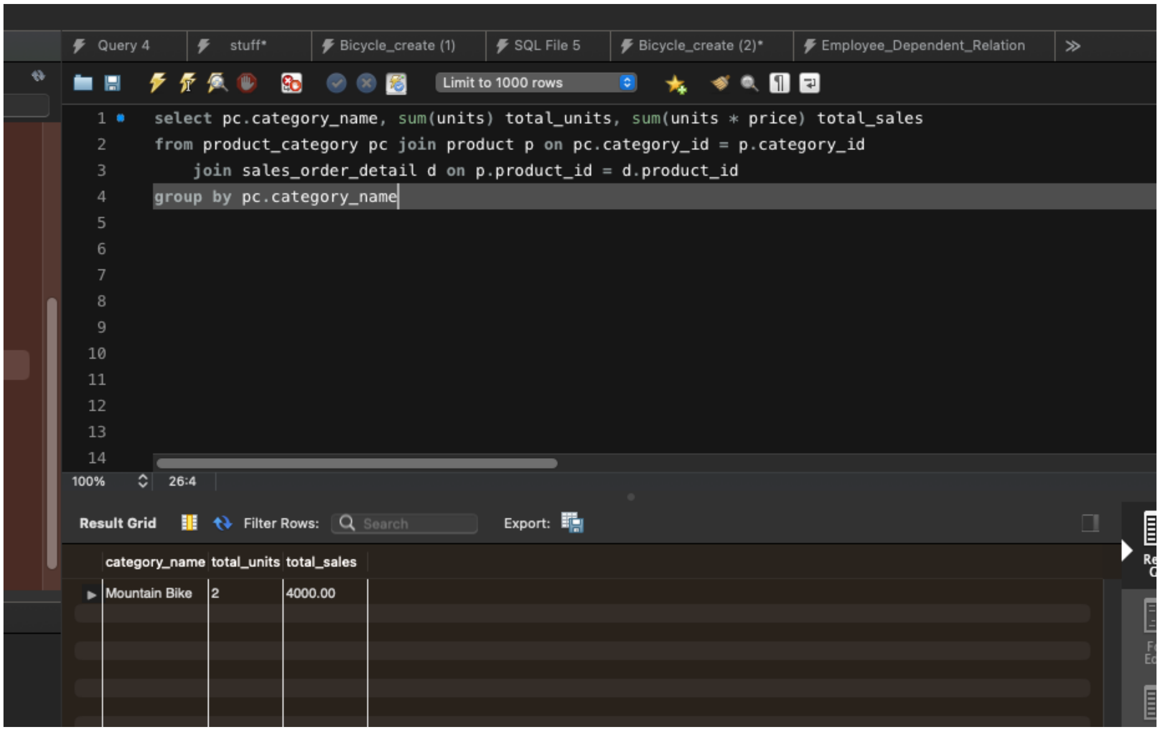
```
with cte_product_total_units as
(
    select p.product_name, sum(d.units) total_units_sold
    from product p
        join sales_order_detail d on p.product_id = d.product_id
    group by p.product_name
)
select c.product_name, c.total_units_sold
from cte_product_total_units c
where c.total_units_sold = (
    select max(total_units_sold) from cte_product_total_units);
```

product_na...	total_units_s...
Z Bike	2

4.3.8 List most popular bicycle category based on sales

- This query will project the category name which has sold the most, determined by sales numbers (units and total sales)

```
select pc.category_name, sum(units) total_units, sum(units * price) total_sales
from product_category pc join product p on pc.category_id = p.category_id
            join sales_order_detail d on p.product_id = d.product_id
group by pc.category_name
```



```
select pc.category_name, sum(units) total_units, sum(units * price) total_sales
from product_category pc join product p on pc.category_id = p.category_id
    join sales_order_detail d on p.product_id = d.product_id
group by pc.category_name
```

category_name	total_units	total_sales
Mountain Bike	2	4000.00

5. CRUD Matrix

5.1 List of Entity Types

E1: Company_Project
E2: Customer
E3: Customer_Payment
E4: Department
E5: Dependent
E6: Employee
E7: Employee_Company_Project
E8: Employee_Dependent_Relation
E9: Job_Title
E10: Payment_Type
E11: Product
E12: Product_Category
E13: Product_Wholesaler
E14: Sales_Order
E15: Sales_Order_Detail
E16: Wholesaler

5.2 List of Functions/Procedures

Please see below as well as appendix for function/procedure statement details

F1: Insert/update/delete/retrieve a Company_Project
F2: Insert/update/delete/retrieve a Customer
F3: Insert/update/delete/retrieve a Customer_Payment
F4: Insert/update/delete/retrieve a Department
F5: Insert/update/delete/retrieve a Dependent
F6: Insert/update/delete/retrieve a Employee
F7: Insert/update/delete/retrieve a Employee_Company_Project
F8: Insert/update/delete/retrieve a Employee_Dependent_Relation
F9: Insert/update/delete/retrieve a Job_Title
F10: Insert/update/delete/retrieve a Payment_Type
F11: Insert/update/delete/retrieve a Product
F12: Insert/update/delete/retrieve a Product_Category
F13: Insert/update/delete/retrieve a Product_Wholesaler
F14: Insert/update/delete/retrieve a Sales_Order
F15: Insert/update/delete/retrieve a Sales_Order_Detail
F16: Insert/update/delete/retrieve a Wholesaler
F17: Retrieve List company Projects and who worked on the Projects
F18: Retrieve Customer Purchase History
F19: Retrieve List employees with their Dependents
F20: Retrieve List all bicycles by category
F21: Retrieve List total sales and total units sold by bicycle Category
F22: Retrieve List best selling bicycles by revenue
F23: Retrieve List best selling bikes by units sold
F24: Retrieve List most popular bicycle category based on sales
F25: Insert/Upload spreadsheet to Employee table in database
F26: Create/Insert new customer into Customer table
F27: Create/Insert new sales order into Sales_Order table
F28: Retrieve and generate error if a customer making purchase has outstanding balance

(Please see Appendix A for statement details (F1 through F16))

5 Functional Requirements from Section 2.3 (Mapped to F26-F30):

- **Upload spreadsheets (CSV) to Employee table in database (or any other table).**

```
BULK INSERT Employee
FROM ' Desktop:\New_Office_Employees_Information.csv’
WITH
(
FIRSTROW = 2, --Second row if header row in file
FIELDTERMINATOR = ',',  --CSV field delimiter
ROWTERMINATOR = '\n',   --Use to shift the control to next row
ERRORFILE = 'error_file_path',
TABLOCK
)
```

- **Create a Customer Account (using procedure p_createCustomer)**

```
DELIMITER $$
CREATE PROCEDURE p_createCustomer (
        IN p_FName VARCHAR(25),
        IN p_LName VARCHAR(25),
        IN p_address VARCHAR(25),
        IN p_city VARCHAR(25),
        IN p_state CHAR(2),
        IN p_zip VARCHAR(5) )
BEGIN
        DECLARE v_custExists INT;

        SELECT COUNT(1) INTO v_custExists
        FROM CUSTOMER
        WHERE FIRST_NAME = p_FName AND LAST_NAME = p_LName
        AND ADDRESS = p_address AND CITY = p_city
        AND STATE = p_state AND ZIP = p_zip;

        IF v_custExists = 0 THEN

                INSERT INTO CUSTOMER (
                        FIRST_NAME, LAST_NAME, ADDRESS, CITY, STATE, ZIP,
                        CURRENT_BALANCE)
                VALUES (p_FName, p_LName, p_address, p_city, p_state, p_zip, 0);

        END IF;
END$$
DELIMITER ;
```

- **Create a Sales Order (using procedure p_createSalesOrders)**

```
DELIMITER $$
CREATE PROCEDURE p_createSalesOrders (
        IN p_employeeID INT,
        IN p_salesOrderDate DATE,
        IN p_custID INT,
        IN p_paymentTypeID INT,
        IN p_productID INT,
        IN p_units INT )
BEGIN

        DECLARE v_salesOrderID INT;
        DECLARE v_custPaymentID INT;
        DECLARE v_price FLOAT(7,2);

        SELECT CUSTOMER_PAYMENT_ID INTO v_custPaymentID
         FROM CUSTOMER_PAYMENT
        WHERE CUSTOMER_ID = p_custID
         AND PAYMENT_TYPE_ID = p_paymentTypeID;

         IF v_custPaymentID IS NULL THEN

                INSERT INTO CUSTOMER_PAYMENT (CUSTOMER_ID, PAYMENT_TYPE_ID)
                VALUES (p_custID, p_paymentTypeID);

        END IF;

        SELECT SALES_ORDER_ID INTO v_salesOrderID
        FROM SALES_ORDER
         WHERE CUSTOMER_ID = p_custID
         AND SALES_ORDER_DATE = p_salesOrderDate
        AND EMPLOYEE_ID = p_employeeID
        AND CUSTOMER_PAYMENT_ID = v_custPaymentID;

        IF v_salesOrderID IS NULL THEN

        INSERT INTO SALES_ORDER (
                CUSTOMER_ID, SALES_ORDER_DATE, CUSTOMER_PAYMENT_ID,
EMPLOYEE_ID)
        VALUES (p_custID, p_salesOrderDate, p_paymentTypeID, p_employeeID);

        END IF;

        SELECT PRODUCT_PRICE INTO v_price
        FROM PRODUCT
        WHERE PRODUCT_ID = p_productID;

        SELECT SALES_ORDER_ID INTO v_salesOrderID
        FROM SALES_ORDER
        WHERE CUSTOMER_ID = p_custID
        AND SALES_ORDER_DATE = p_salesOrderDate
        AND EMPLOYEE_ID = p_employeeID
        AND CUSTOMER_PAYMENT_ID = v_custPaymentID;

        INSERT INTO SALES_ORDER_DETAIL (
                SALES_ORDER_ID, PRODUCT_ID, PRICE, UNITS)
        VALUES (v_salesOrderID, p_productID, v_price, p_units);

END$$
DELIMITER ;
```

- **Keep track of employees (using function f_getEmpDept)**

```
DELIMITER $$
CREATE FUNCTION f_getEmpDept ( p_EmpID INT ) RETURNS VARCHAR(25)
DETERMINISTIC

BEGIN

        DECLARE v_deptName VARCHAR(25);

        SELECT DEPARTMENT_NAME INTO v_deptName
        FROM DEPARTMENT D JOIN EMPLOYEE E ON D.DEPARTMENT_ID =
        E.DEPARTMENT_ID
        WHERE E.EMPLOYEE_ID = p_empID;

        RETURN v_deptName;

END$$
DELIMITER;
```

- Track payments (create error/flag if customer trys to make a purchase and has an outstanding balance).

```
DELIMITER $$
CREATE TRIGGER chk_custBalance BEFORE INSERT ON sales_order
FOR EACH ROW
BEGIN
        DECLARE v_custBalance FLOAT(7,2);

        SELECT CURRENT_BALANCE INTO v_custBalance
        FROM CUSTOMER
        WHERE CUSTOMER_ID = NEW.CUSTOMER_ID;

        IF v_custBalance > 0 THEN
                SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Customer has a balance.
        Cannot create sales order.';
        END IF;

END;$$
DELIMITER ;
```

Function/Entity Interaction	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16
F1	CRUD															
F2		CRUD														
F3			CRUD													
F4				CRUD												
F5					CRUD											
F6						CRUD										
F7							CRUD									
F8								CRUD								
F9									CRUD							
F10										CRUD						
F11											CRUD					
F12												CRUD				
F13													CRUD			
F14														CRUD		
F15															CRUD	
F16																CRUD
F17	R					R	R									
F18		R	R							R	R			R	R	
F19					R	R		R								
F20											R	R				
F21											R	R			R	
F22											R				R	
F23											R				R	
F24											R	R			R	
F25						C										
F26		C														
F27														C		
F28		R												R		

6. Concluding Remarks

Given that I had no prior experience with database design or implementation prior to this class, I feel as though I learned an enormous amount through building and designing the Bent Spoke Bike Store Database. The most important lesson I learned from completing this project was to make sure to spend time upfront to really "flesh out" and understand how the database will be used by the end user. I spent a fair amount of time trying to ensure that the functionality surrounding sales information would be able to be retrieved in the manner I had envisioned and in retrospect I feel as though I wouldn't have had to spend as much time on this portion of the project had I started with a more holistic and thought-out initial design.

I feel as though the strengths of this project were some of my more complex database queries (such as my Customer Purchase History query which links 6 tables) as well as some of my advanced features. My "stand out" advanced features include the trigger for my sales_order_detail table which will automatically decrement the units_in_stock in the product table upon each additional bike sold. My "stand out" stored procedure is able to show all sales orders by customer. I feel particularly proud of these two features because I came close to giving up on them due to the complexity but persevered and managed to get them working prior to the assignment deadline.

I feel as though the weakness of this assignment was likely the simplicity of a standard retail business, so this type of model likely won't impress the reader. I had initially wanted to build a more sophisticated database to model several machine learning research processes at an investment firm, but I had worried from the start that proceeding down that path as a first database design project would be setting myself up for failure as the complexity of such a project would have increased exponentially when attempting to meet all requirements listed on this document.

If I had more time to work on this project, I would further build out the sophistication to more fully resemble a real-world business. I would first build out the company's financial system starting with accounting related entities including salaries, sales bonus, and overtime attributes. I would include record keeping for employee payment as well as include social security numbers for all employees for keeping track (and potentially withholding taxes for them). By also adding the employee SSN I would also be able to have 2 employees by the same name, which I am not currently able to do with the current database structure.

Additionally, I worry that I may have spent far too much time focusing on the Advanced Features section, which at the time I felt was the correct path to ensure they worked properly but I now fear this may have detracted from the quality of the rest of the assignment. If I had more time, I would enhance the rest of the assignment more and focus less on the Advanced Features.

Appendices

Note: if any ddl, insert, or other statements are missing from below please check the appropriate section above. I didn't want to double count statements which would make the writeup document overly long resulting in potential problems during submission.

Appendix A - DDL, INSERT, SELECT Statements

Implemented Statements For Tables and Data Input:

```
-- Table: Company_Project
CREATE TABLE Company_Project (
   Project_ID int NOT NULL,
   Project_Name varchar(25) NOT NULL,
   CONSTRAINT Company_Project_pk PRIMARY KEY (Project_ID)
);

-- Table: Customer
CREATE TABLE Customer (
   Customer_ID int NOT NULL auto_increment,
   First_Name varchar(25) NOT NULL,
   Last_Name varchar(25) NOT NULL,
   Address varchar(25) NOT NULL,
   City varchar(25) NOT NULL,
   State char(2) NOT NULL,
   Zip varchar(5) NOT NULL,
   Current_Balance float(7,2) NOT NULL,
   CONSTRAINT Customer_pk PRIMARY KEY (Customer_ID)
);

-- Table: Customer_Payment
CREATE TABLE Customer_Payment (
   Customer_Payment_ID int NOT NULL auto_increment,
   Customer_ID int NOT NULL,
   Payment_Type_ID int NOT NULL,
   CONSTRAINT Customer_Payment_pk PRIMARY KEY (Customer_Payment_ID)
);

-- Table: Department
CREATE TABLE Department (
   Department_ID int NOT NULL,
   Department_Name varchar(25) NOT NULL,
   CONSTRAINT Department_pk PRIMARY KEY (Department_ID)
);
```

```
-- Table: Dependent
CREATE TABLE Dependent (
   Dependent_ID int NOT NULL,
   Employee_ID int NOT NULL,
   First_Name varchar(25) NOT NULL,
   Last_Name varchar(25) NOT NULL,
   Employee_Dependent_Relation_ID int NOT NULL,
   CONSTRAINT Dependent_pk PRIMARY KEY (Dependent_ID)
);

-- Table: Employee
CREATE TABLE Employee (
   Employee_ID int NOT NULL,
   First_Name varchar(25) NOT NULL,
   Last_Name varchar(25) NOT NULL,
   Job_Title_ID int NOT NULL,
   DOB date NOT NULL,
   Hire_Date date NOT NULL,
   Primary_Phone varchar(10) NOT NULL,
   Supervisor_Employee_ID int NULL,
   Department_ID int NOT NULL,
   CONSTRAINT Employee_pk PRIMARY KEY (Employee_ID)
);

-- Table: Employee_Company_Project
CREATE TABLE Employee_Company_Project (
   Employee_ID int NOT NULL,
   Project_ID int NOT NULL,
   CONSTRAINT Employee_Company_Project_pk PRIMARY KEY
(Employee_ID,Project_ID)
);

-- Table: Employee_Dependent_Relation
CREATE TABLE Employee_Dependent_Relation (
   Employee_Dependent_Relation_ID int NOT NULL,
   Employee_Dependent_Relation_Name varchar(25) NOT NULL,
   CONSTRAINT Employee_Dependent_Relation_pk PRIMARY KEY
(Employee_Dependent_Relation_ID)
);

-- Table: Job_Title
CREATE TABLE Job_Title (
   Job_Title_ID int NOT NULL,
   Job_Title_Name varchar(25) NOT NULL,
   CONSTRAINT Job_Title_pk PRIMARY KEY (Job_Title_ID)
);

-- Table: Payment_Type
CREATE TABLE Payment_Type (
   Payment_Type_ID int NOT NULL,
   Payment_Type_Name varchar(25) NOT NULL,
   CONSTRAINT Payment_Type_pk PRIMARY KEY (Payment_Type_ID)
);

-- Table: Product
CREATE TABLE Product (
   Product_ID int NOT NULL,
   Category_ID int NOT NULL,
   Product_Name varchar(25) NOT NULL,
   Product_Price float(7,2) NOT NULL,
   Units_In_Stock int NOT NULL,
   CONSTRAINT Product_pk PRIMARY KEY (Product_ID)
);

-- Table: Product_Category
CREATE TABLE Product_Category (
   Category_ID int NOT NULL,
   Category_Name varchar(25) NOT NULL,
   CONSTRAINT Product_Category_pk PRIMARY KEY (Category_ID)
);

-- Table: Product_Wholesaler
CREATE TABLE Product_Wholesaler (
   Product_ID int NOT NULL,
   Wholesaler_ID int NOT NULL,
   WSP float(7,2) NOT NULL,
   CONSTRAINT Product_Wholesaler_pk PRIMARY KEY
(Product_ID,Wholesaler_ID)
);

-- Table: Sales_Order
CREATE TABLE Sales_Order (
   Sales_Order_ID int NOT NULL auto_increment,
   Customer_ID int NOT NULL,
   Sales_Order_Date date NOT NULL,
   Customer_Payment_ID int NOT NULL,
   Employee_ID int NOT NULL,
   CONSTRAINT Sales_Order_pk PRIMARY KEY (Sales_Order_ID)
);

-- Table: Sales_Order_Detail
```

```
CREATE TABLE Sales_Order_Detail (
   Sales_Order_Detail_ID int NOT NULL auto_increment,
         Sales_Order_ID int NOT NULL,
   Product_ID int NOT NULL,
   Price float(7,2) NOT NULL,
   Units int NOT NULL,
   CONSTRAINT Sales_Order_Detail_pk PRIMARY KEY (Sales_Order_Detail_ID)
);

-- Table: Wholesaler
CREATE TABLE Wholesaler (
   Wholesaler_ID int NOT NULL,
   Wholesaler_Name varchar(25) NOT NULL,
   Contact varchar(25) NOT NULL,
   Phone varchar(10) NOT NULL,
   Email varchar(25) NOT NULL,
   CONSTRAINT Wholesaler_pk PRIMARY KEY (Wholesaler_ID)
);

-- foreign keys
-- Reference: Customer_Payment_Customer (table: Customer_Payment)
ALTER TABLE Customer_Payment ADD CONSTRAINT
Customer_Payment_Customer FOREIGN KEY Customer_Payment_Customer
(Customer_ID)
   REFERENCES Customer (Customer_ID);

-- Reference: Customer_Payment_Payment_Type (table: Customer_Payment)
ALTER TABLE Customer_Payment ADD CONSTRAINT
Customer_Payment_Payment_Type FOREIGN KEY
Customer_Payment_Payment_Type (Payment_Type_ID)
   REFERENCES Payment_Type (Payment_Type_ID);

-- Reference: Dependent_Employee (table: Dependent)
ALTER TABLE Dependent ADD CONSTRAINT Dependent_Employee FOREIGN
KEY Dependent_Employee (Employee_ID)
   REFERENCES Employee (Employee_ID);

-- Reference: Dependent_Employee_Dependent_Relation (table: Dependent)
ALTER TABLE Dependent ADD CONSTRAINT
Dependent_Employee_Dependent_Relation FOREIGN KEY
Dependent_Employee_Dependent_Relation (Employee_Dependent_Relation_ID)
   REFERENCES Employee_Dependent_Relation (Employee_Dependent_Relation_ID);

-- Reference: Employee_Company_Project_Company_Project (table:
Employee_Company_Project)
```

```
ALTER TABLE Employee_Company_Project ADD CONSTRAINT
Employee_Company_Project_Company_Project FOREIGN KEY
Employee_Company_Project_Company_Project (Project_ID)
   REFERENCES Company_Project (Project_ID);

-- Reference: Employee_Company_Project_Employee (table:
Employee_Company_Project)
ALTER TABLE Employee_Company_Project ADD CONSTRAINT
Employee_Company_Project_Employee FOREIGN KEY
Employee_Company_Project_Employee (Employee_ID)
   REFERENCES Employee (Employee_ID);

-- Reference: Employee_Department (table: Employee)
ALTER TABLE Employee ADD CONSTRAINT Employee_Department FOREIGN
KEY Employee_Department (Department_ID)
   REFERENCES Department (Department_ID);

-- Reference: Employee_Employee (table: Employee)
ALTER TABLE Employee ADD CONSTRAINT Employee_Employee FOREIGN KEY
Employee_Employee (Supervisor_Employee_ID)
   REFERENCES Employee (Employee_ID);

-- Reference: Employee_Job_Title (table: Employee)
ALTER TABLE Employee ADD CONSTRAINT Employee_Job_Title FOREIGN KEY
Employee_Job_Title (Job_Title_ID)
   REFERENCES Job_Title (Job_Title_ID);

-- Reference: Product_Product_Category (table: Product)
ALTER TABLE Product ADD CONSTRAINT Product_Product_Category FOREIGN
KEY Product_Product_Category (Category_ID)
   REFERENCES Product_Category (Category_ID);

-- Reference: Product_Wholesaler_Product (table: Product_Wholesaler)
ALTER TABLE Product_Wholesaler ADD CONSTRAINT
Product_Wholesaler_Product FOREIGN KEY Product_Wholesaler_Product
(Product_ID)
   REFERENCES Product (Product_ID);

-- Reference: Product_Wholesaler_Wholesaler (table: Product_Wholesaler)
ALTER TABLE Product_Wholesaler ADD CONSTRAINT
Product_Wholesaler_Wholesaler FOREIGN KEY Product_Wholesaler_Wholesaler
(Wholesaler_ID)
   REFERENCES Wholesaler (Wholesaler_ID);

-- Reference: Sales_Order_Customer (table: Sales_Order)
```

```
ALTER TABLE Sales_Order ADD CONSTRAINT Sales_Order_Customer FOREIGN
KEY Sales_Order_Customer (Customer_ID)
   REFERENCES Customer (Customer_ID);

-- Reference: Sales_Order_Customer_Payment (table: Sales_Order)
ALTER TABLE Sales_Order ADD CONSTRAINT Sales_Order_Customer_Payment
FOREIGN KEY Sales_Order_Customer_Payment (Customer_Payment_ID)
   REFERENCES Customer_Payment (Customer_Payment_ID);

-- Reference: Sales_Order_Detail_Product (table: Sales_Order_Detail)
ALTER TABLE Sales_Order_Detail ADD CONSTRAINT Sales_Order_Detail_Product
FOREIGN KEY Sales_Order_Detail_Product (Product_ID)
   REFERENCES Product (Product_ID);

-- Reference: Sales_Order_Detail_Sales_Order (table: Sales_Order_Detail)
ALTER TABLE Sales_Order_Detail ADD CONSTRAINT
Sales_Order_Detail_Sales_Order FOREIGN KEY Sales_Order_Detail_Sales_Order
(Sales_Order_ID)
   REFERENCES Sales_Order (Sales_Order_ID);

-- Reference: Sales_Order_Employee (table: Sales_Order)
ALTER TABLE Sales_Order ADD CONSTRAINT Sales_Order_Employee FOREIGN
KEY Sales_Order_Employee (Employee_ID)
   REFERENCES Employee (Employee_ID);
```

Create/Insert Statements

Job_Title:

```
INSERT INTO `bike_store_final`.`Job_Title` (`Job_Title_ID`, `Job_Title_Name`)
VALUES ('1', 'Sales Manager');
INSERT INTO `bike_store_final`.`Job_Title` (`Job_Title_ID`, `Job_title_name`)
VALUES ('2', 'Sales Rep');
INSERT INTO `bike_store_final`.`Job_Title` (`Job_Title_ID`, `Job_Title_Name`)
VALUES ('3', 'Store Manager');
INSERT INTO `bike_store_final`.`Job_Title` (`Job_Title_ID`, `Job_Title_Name`)
VALUES ('4', 'Mechanic');
INSERT INTO `bike_store_final`.`Job_Title` (`Job_Title_ID`, `Job_Title_Name`)
VALUES ('5', 'Events Coordinator');
INSERT INTO `bike_store_final`.`Job_Title` (`Job_Title_ID`, `Job_Title_Name`)
VALUES ('6', 'Territory Manager');
INSERT INTO `bike_store_final`.`Job_Title` (`Job_Title_ID`, `Job_Title_Name`)
VALUES ('7', 'IT Analyst');
```

Department:

```
INSERT INTO `bike_store_final`.`Department` (`Department_ID`, `Department_Name`)
VALUES ('1', 'Sales');
```

```
INSERT INTO `bike_store_final`.`Department` (`Department_ID`, `Department_Name`)
VALUES ('2', 'Events');
INSERT INTO `bike_store_final`.`Department` (`Department_ID`, `Department_Name`)
VALUES ('3', 'Management');
INSERT INTO `bike_store_final`.`Department` (`Department_ID`, `Department_Name`)
VALUES ('4', 'Mechanical');
INSERT INTO `bike_store_final`.`Department` (`Department_ID`, `Department_Name`)
VALUES ('5', 'Information Technology');
```

Company_Project:

```
INSERT INTO `bike_store_final`.`Company_Project` (`Project_ID`, `Project_Name`)
VALUES ('1', 'Build New Website');
INSERT INTO `bike_store_final`.`Company_Project` (`Project_ID`, `Project_Name`)
VALUES ('2', 'Build NV Store');
INSERT INTO `bike_store_final`.`Company_Project` (`Project_ID`, `Project_Name`)
VALUES ('3', 'Hire Employees for NY');
```

Payment_Type:

```
INSERT INTO `bike_store_final`.`Payment_Type` (`Payment_Type_ID`,
`Payment_Type_Name`) VALUES ('1', 'Cash');
INSERT INTO `bike_store_final`.`Payment_Type` (`Payment_Type_ID`,
`Payment_Type_Name`) VALUES ('2', 'Credit Card');
INSERT INTO `bike_store_final`.`Payment_Type` (`Payment_Type_ID`,
`Payment_Type_Name`) VALUES ('3', 'Check');
INSERT INTO `bike_store_final`.`Payment_Type` (`Payment_Type_ID`,
`Payment_Type_Name`) VALUES ('4', 'Crypto Currency');
```

Employee_Dependent_Relation:

```
INSERT INTO `bike_store_final`.`Employee_Dependent_Relation`
(`Employee_Dependent_Relation_ID`, `Employee_Dependent_Relation_Name`)
VALUES ('1', 'Daughter');
INSERT INTO `bike_store_final`.`Employee_Dependent_Relation`
(`Employee_Dependent_Relation_ID`, `Employee_Dependent_Relation_Name`)
VALUES ('2', 'Son');
INSERT INTO `bike_store_final`.`Employee_Dependent_Relation`
(`Employee_Dependent_Relation_ID`, `Employee_Dependent_Relation_Name`)
VALUES ('3', 'Stepchild');
INSERT INTO `bike_store_final`.`Employee_Dependent_Relation`
(`Employee_Dependent_Relation_ID`, `Employee_Dependent_Relation_Name`)
VALUES ('4', 'Grandson');
INSERT INTO `bike_store_final`.`Employee_Dependent_Relation`
(`Employee_Dependent_Relation_ID`, `Employee_Dependent_Relation_Name`)
VALUES ('5', 'Granddaughter');
INSERT INTO `bike_store_final`.`Employee_Dependent_Relation`
(`Employee_Dependent_Relation_ID`, `Employee_Dependent_Relation_Name`)
VALUES ('6', 'Brother');
```

```
INSERT INTO `bike_store_final`.`Employee_Dependent_Relation`
(`Employee_Dependent_Relation_ID`, `Employee_Dependent_Relation_Name`)
VALUES ('7', 'Sister');
```

Product_Category:

```
INSERT INTO `bike_store_final`.`Product_Category` (`Category_ID`,
`Category_Name`) VALUES ('1', 'Mountain Bike');
INSERT INTO `bike_store_final`.`Product_Category` (`Category_ID`,
`Category_Name`) VALUES ('2', 'Road Bike');
```

Wholesaler:

```
INSERT INTO `bike_store_final`.`Wholesaler` (`Wholesaler_ID`, `Wholesaler_Name`,
`Contact`, `Phone`, `Email`) VALUES ('1', 'ABC', 'James', '8974562345',
'James@abc.com');
INSERT INTO `bike_store_final`.`Wholesaler` (`Wholesaler_ID`, `Wholesaler_Name`,
`Contact`, `Phone`, `Email`) VALUES ('2', 'DEF', 'Jill', '5104984576', 'jill@def.com');
INSERT INTO `bike_store_final`.`Wholesaler` (`Wholesaler_ID`, `Wholesaler_Name`,
`Contact`, `Phone`, `Email`) VALUES ('3', 'GHI', 'Omar', '4156453489',
'omar@ghi.com');
```

Product:

```
INSERT INTO `bike_store_final`.`Product` (`Product_ID`, `Category_ID`,
`Product_Name`, `Product_Price`, `Units_In_Stock`) VALUES ('1', '1', 'Z Bike', '2000',
'20');
INSERT INTO `bike_store_final`.`Product` (`Product_ID`, `Category_ID`,
`Product_Name`, `Product_Price`, `Units_In_Stock`) VALUES ('2', '2', 'A Bike', '2500',
'3');
INSERT INTO `bike_store_final`.`Product` (`Product_ID`, `Category_ID`,
`Product_Name`, `Product_Price`, `Units_In_Stock`) VALUES ('3', '1', 'X Bike', '3000',
'19');
INSERT INTO `bike_store_final`.`Product` (`Product_ID`, `Category_ID`,
`Product_Name`, `Product_Price`, `Units_In_Stock`) VALUES ('4', '1', 'B Bike', '1500',
'25');
INSERT INTO `bike_store_final`.`Product` (`Product_ID`, `Category_ID`,
`Product_Name`, `Product_Price`, `Units_In_Stock`) VALUES ('5', '1', 'C Bike', '3500',
'30');
```

Customer:

```
INSERT INTO `bike_store_final`.`Customer` (`Customer_ID`, `First_Name`,
`Last_Name`, `Address`, `City`, `State`, `Zip`, `Current_Balance`) VALUES ('1', 'Jeff',
'Roland', '11 Pine Rd.', 'Buffalo', 'NY', '85769', '0');
INSERT INTO `bike_store_final`.`Customer` (`Customer_ID`, `First_Name`,
`Last_Name`, `Address`, `City`, `State`, `Zip`, `Current_Balance`) VALUES ('2', 'Bill',
'Riley', '3 Oak Ln.', 'Oakland', 'CA', '95768', '1000');
```

```
INSERT INTO `bike_store_final`.`Customer` (`Customer_ID`, `First_Name`, `Last_Name`, `Address`, `City`, `State`, `Zip`, `Current_Balance`) VALUES ('3', 'Mary', 'Kay', '56 First St.', 'Larkspur', 'CA', '93457', '0');
INSERT INTO `bike_store_final`.`Customer` (`Customer_ID`, `First_Name`, `Last_Name`, `Address`, `City`, `State`, `Zip`, `Current_Balance`) VALUES ('4', 'Jamal', 'Bash', '100 Jasper Rd.', 'Chico', 'CA', '14325', '75');
INSERT INTO `bike_store_final`.`Customer` (`Customer_ID`, `First_Name`, `Last_Name`, `Address`, `City`, `State`, `Zip`, `Current_Balance`) VALUES ('5', 'Xi', 'Chen', '3 Maple Ln.', 'Sacramento', 'CA', '96754', '0');
INSERT INTO `bike_store_final`.`Customer` (`Customer_ID`, `First_Name`, `Last_Name`, `Address`, `City`, `State`, `Zip`, `Current_Balance`) VALUES ('6', 'James', 'Kerr', '5 James Rd', 'LA', 'CA', '45637', '67');
```

Employee:

```
INSERT INTO `bike_store_final`.`Employee` (`Employee_ID`, `First_Name`, `Last_Name`, `Job_Title_ID`, `DOB`, `Hire_Date`, `Primary_Phone`, `Supervisor_Employee_ID`, `Department_ID`) VALUES ('2', 'Karen', 'Will', '3', '1978-10-05', '2019-03-20', '4578493029', NULL,'3');
INSERT INTO `bike_store_final`.`Employee` (`Employee_ID`, `First_Name`, `Last_Name`, `Job_Title_ID`, `DOB`, `Hire_Date`, `Primary_Phone`, `Supervisor_Employee_ID`, `Department_ID`) VALUES ('3', 'Ken', 'Bird', '2', '1980-04-05', '2009-04-20', '5678349230', '2', '1');
INSERT INTO `bike_store_final`.`Employee` (`Employee_ID`, `First_Name`, `Last_Name`, `Job_Title_ID`, `DOB`, `Hire_Date`, `Primary_Phone`, `Supervisor_Employee_ID`, `Department_ID`) VALUES ('4', 'Sarah', 'Klein', '4', '1981-05-06', '2014-02-05', '5678345230', '2', '4');
INSERT INTO `bike_store_final`.`Employee` (`Employee_ID`, `First_Name`, `Last_Name`, `Job_Title_ID`, `DOB`, `Hire_Date`, `Primary_Phone`, `Supervisor_Employee_ID`, `Department_ID`) VALUES ('5', 'Corbit', 'Kru', '5', '1982-07-10', '2018-04-06', '9668345231', '2', '2');
INSERT INTO `bike_store_final`.`Employee` (`Employee_ID`, `First_Name`, `Last_Name`, `Job_Title_ID`, `DOB`, `Hire_Date`, `Primary_Phone`, `Supervisor_Employee_ID`, `Department_ID`) VALUES ('6', 'Riley', 'Barns', '2', '1984-08-12', '2019-05-10', '1678359231', '2', '1');
```

Emloyee_Company_Project:

```
INSERT INTO `bike_store_final`.`Employee_Company_Project` (`Employee_ID`, `Project_ID`) VALUES ('2', '2');
```

Dependent:

```
INSERT INTO `bike_store_final`.`Dependent` (`Dependent_ID`, `Employee_ID`, `First_Name`, `Last_Name`, `Employee_Dependent_Relation_ID`) VALUES ('1', '2', 'Jake', 'Will', '2');
INSERT INTO `bike_store_final`.`Dependent` (`Dependent_ID`, `Employee_ID`, `First_Name`, `Last_Name`, `Employee_Dependent_Relation_ID`) VALUES ('2', '3', 'Erin', 'Bird', '1');
```

Product_Wholesaler:

```
INSERT INTO `bike_store_final`.`Product_Wholesaler` (`Product_ID`, `Wholesaler_ID`, `WSP`) VALUES ('1', '1', '1000');
INSERT INTO `bike_store_final`.`Product_Wholesaler` (`Product_ID`, `Wholesaler_ID`, `WSP`) VALUES ('2', '3', '1500');
```

Customer_Payment:

```
INSERT INTO `bike_store_final`.`Customer_Payment` (`Customer_Payment_ID`, `Customer_ID`, `Payment_Type_ID`) VALUES ('1', '1', '1');
INSERT INTO `bike_store_final`.`Customer_Payment` (`Customer_Payment_ID`, `Customer_ID`, `Payment_Type_ID`) VALUES ('2', '2', '3');
INSERT INTO `bike_store_final`.`Customer_Payment` (`Customer_Payment_ID`, `Customer_ID`, `Payment_Type_ID`) VALUES ('3', '3', '2');
```

Sale_Order:

```
INSERT INTO `bike_store_final`.`Sales_Order` (`Sales_Order_ID`, `Customer_ID`, `Sales_Order_Date`, `Customer_Payment_ID`, `Employee_ID`) VALUES ('1', '1', '2022-11-01', '1', '6');
INSERT INTO `bike_store_final`.`Sales_Order` (`Sales_Order_ID`, `Customer_ID`, `Sales_Order_Date`, `Customer_Payment_ID`, `Employee_ID`) VALUES ('2', '3', '2022-11-08', '3', '3');
INSERT INTO `bike_store_final`.`Sales_Order` (`Sales_Order_ID`, `Customer_ID`, `Sales_Order_Date`, `Customer_Payment_ID`, `Employee_ID`) VALUES ('3', '2', '2022-11-08', '2', '3');
INSERT INTO `bike_store_final`.`Sales_Order` (`Sales_Order_ID`, `Customer_ID`, `Sales_Order_Date`, `Customer_Payment_ID`, `Employee_ID`) VALUES ('4', '4', '2022-11-10', '1', '6');
```

Sales_Order_Detail:

```
INSERT INTO `bike_store_final`.`Sales_Order_Detail` (`Sales_Order_Detail_ID`, `Sales_Order_ID`, `Product_ID`, `Price`, `Units`) VALUES ('1', '1', '1', '2000', '2');
```

Implemented Functions for CRUD Table:

F1: Insert/update/delete/retrieve a Company_Project

Create:

```
INSERT INTO `bike_store_final`.`Company_Project` (`Project_ID`, `Project_Name`) VALUES ('4', 'Hire Employees for TX');
```

Read:

```
SELECT *
FROM Company_Project
```

Update:

```
UPDATE Company_Project
SET Project_Name = "Hire Employee for MO"
WHERE Project_ID = 4, SSN = '4984564512'
```

Delete:

```
DELETE FROM Company_Project
WHERE Project_ID = 4
```

F2: Insert/update/delete/retrieve a Customer

Create:

```
INSERT INTO `bike_store_final`.`Customer` (`Customer_ID`, `First_Name`,
`Last_Name`, `Address`, `City`, `State`, `Zip`, `Current_Balance`) VALUES ('50', 'Jan',
'Fran', '10 Real Rd.', 'Harrisburg', 'PA', '85769', '0');
```

Read:

```
SELECT *
FROM Customer
```

Update:

```
UPDATE Customer
SET City = "Allentown"
WHERE Customer_ID = 50, SSN = '4984564512'
```

Delete:

```
DELETE FROM Customer
WHERE Customer_ID = 50
```

F3: Insert/update/delete/retrieve a Customer_Payment

Create:

```
INSERT INTO `bike_store_final`.`Customer_Payment` (`Customer_Payment_ID`,
`Customer_ID`, `Payment_Type_ID`) VALUES ('10', '1', '1');
```

Read:

```
SELECT *
FROM Customer_Payment
```

Update:

```
UPDATE Customer_Payment
SET Payment_Type_ID = "2"
WHERE Customer_Payment_ID = 10, SSN = '4984564512'
```

Delete:

```
DELETE FROM Customer_Payment
WHERE Customer_Payment_ID = 10
```

F4: Insert/update/delete/retrieve a Department

Create:

```
INSERT INTO `bike_store_final`.`Department` (`Department_ID`, `Department_Name`)
VALUES ('11', 'Manufacturing');
```

Read:

```
SELECT *
FROM Department
```

Update:

```
UPDATE Department
SET Payment_Type_ID = "Bike Manufacturing"
WHERE Department_ID = 11, SSN = '4984564512'
```

Delete:

```
DELETE FROM Department
WHERE Department_ID = 11
```

F5: Insert/update/delete/retrieve a Dependent

Create:

```
INSERT INTO `bike_store_final`.`Dependent` (`Dependent_ID`, `Employee_ID`,
`First_Name`, `Last_Name`, `Employee_Dependent_Relation_ID`) VALUES ('21', '2',
'James', 'Will', '1');
```

Read:

```
SELECT *
FROM Dependent
```

Update:

```
UPDATE Dependent
SET First_Name = "Jamie"
WHERE Dependent_ID = 21, SSN = '4984564512'
```

Delete:

```
DELETE FROM Dependent
WHERE Dependent_ID = 21
```

F6: Insert/update/delete/retrieve a Employee

Create:

```
INSERT INTO `bike_store_final`.`Employee` (`Employee_ID`, `First_Name`, `Last_Name`,
`Job_Title_ID`, `DOB`, `Hire_Date`, `Primary_Phone`, `Supervisor_Employee_ID`, `Department_ID`)
VALUES ('10', 'Karl', 'Wims', '3', '1976-10-10', '2019-03-20', '4578493029', NULL,'3');
```

Read:

```
SELECT *
FROM Employee
```

Update:

```
UPDATE Employee
SET Hire_Date = "2020-01-01"
WHERE Employee_ID = 10, SSN = '4984564512'
```

Delete:

```
DELETE FROM Employee
WHERE Employee_ID = 10,
```

F7: Insert/update/delete/retrieve a Employee_Company_Project

Create:

```
INSERT INTO `bike_store_final`.`Employee_Company_Project` (`Employee_ID`, `Project_ID`) VALUES
('3', '2');
```

Read:

```
SELECT *
FROM Employee_Company_Project
```

Update:

```
UPDATE Employee_Company_Project
SET Project_ID = "1"
WHERE Employee_ID = 3, SSN = '4984564512'
```

Delete:

```
DELETE FROM Employee_Company_Project
WHERE Employee_ID = 3
```

F8: Insert/update/delete/retrieve a Employee_Dependent_Relation

Create:

```
INSERT INTO `bike_store_final`.`Employee_Dependent_Relation` (`Employee_Dependent_Relation_ID`,
`Employee_Dependent_Relation_Name`) VALUES ('8', 'Father');
```

Read:

```
SELECT *
```

```
FROM Employee_Dependent_Relation
```

Update:

```
UPDATE Employee_Dependent_Relation
SET Employee_Dependent_Relation_Name = “Mother”
WHERE Employee_Dependent_Relation_ID = 8, SSN = '4984564512'
```

Delete:

```
DELETE FROM Employee_Dependent_Relation
WHERE Employee_Dependent_Relation_ID = 8
```

F9: Insert/update/delete/retrieve a Job_Title

Create:

```
INSERT INTO `bike_store_final`.`Job_Title` (`Job_Title_ID`, `Job_Title_Name`) VALUES ('10', 'CFO');
```

Read:

```
SELECT *
FROM Job_Title
```

Update:

```
UPDATE Job_Title
SET Job_Title_Name = "CEO"
WHERE Job_Title_ID = 10SSN = '4984564512'
```

Delete:

```
DELETE FROM Job_Title
WHERE Job_Title_ID = 10
```

F10: Insert/update/delete/retrieve a Payment_Type

Create:

```
INSERT INTO `bike_store_final`.`Payment_Type` (`Payment_Type_ID`, `Payment_Type_Name`)
VALUES ('5', 'Foreign Currency');
```

Read:

```
SELECT *
FROM Payment_Type
```

Update:

```
UPDATE Payment_Type
SET Payment_Type_Name = "Euros"
WHERE Payment_Type_ID = 5SSN = '4984564512'
```

Delete:

```
DELETE FROM Payment_Type
WHERE Payment_Type_ID = 5SSN
```

F11: Insert/update/delete/retrieve a Product

Create:

```
INSERT INTO `bike_store_final`.`Product` (`Product_ID`, `Category_ID`, `Product_Name`,
`Product_Price`, `Units_In_Stock`) VALUES ('10', '1', 'H Bike', '8000', '20');
```

Read:

```
SELECT *
FROM Product
```

Update:

```
UPDATE Product
SET Product_Name = "Y Bike"
WHERE Product_ID = 10SSN = '4984564512'
```

Delete:

```
DELETE FROM Product
WHERE Product_ID = 10SSN
```

F12: Insert/update/delete/retrieve a Product_Category

Create:

```
INSERT INTO `bike_store_final`.`Product_Category` (`Category_ID`, `Category_Name`) VALUES ('3', 'E
Bike');
```

Read:

```
SELECT *
FROM Product_Category
```

Update:

```
UPDATE Product_Category
SET Category_Name = "Electric Bike"
WHERE Category_ID = 3SSN = '4984564512'
```

Delete:

```
DELETE FROM Product_Category
WHERE Category_ID = 3SSN
```

F13: Insert/update/delete/retrieve a Product_Wholesaler

Create:

```
INSERT INTO `bike_store_final`.`Product_Wholesaler` (`Product_ID`, `Wholesaler_ID`, `WSP`)
VALUES ('3', '1', '3000');
```

Read:

```
SELECT *
FROM Product_Wholesaler
```

Update:

```
UPDATE Product_Wholesaler
SET Wholesaler_ID = "2"
WHERE Product_ID = 3SSN = '4984564512'
```

Delete:

```
DELETE FROM Product_Wholesaler
WHERE Product_ID = 3SSN
```

F14: Insert/update/delete/retrieve a Sales_Order

Create:

```
INSERT INTO `bike_store_final`.`Sales_Order` (`Sales_Order_ID`, `Customer_ID`, `Sales_Order_Date`,
`Customer_Payment_ID`, `Employee_ID`) VALUES ('10', '1', '2021-11-01', '1', '6');
```

Read:

```
SELECT *
FROM Sales_Order
```

Update:

```
UPDATE Sales_Order
SET Sales_Order_Date = '2021-05-11’
WHERE Sales_Order_ID = 10SSN = '4984564512'
```

Delete:

```
DELETE FROM Sales_Order
WHERE Sales_Order_ID = 3SSN
```

F15: Insert/update/delete/retrieve a Sales_Order_Detail

Create:

```
INSERT INTO `bike_store_final`.`Sales_Order_Detail` (`Sales_Order_Detail_ID`, `Sales_Order_ID`,
`Product_ID`, `Price`, `Units`) VALUES ('10', '1', '1', '2000', '2');
```

Read:

```
SELECT *
FROM Sales_Order_Detail
```

Update:

UPDATE Sales_Order_Detail
SET Units = 4
WHERE Sales_Order_Detail_ID = 10= '4984564512'

Delete:

DELETE FROM Sales_Order_Detail
WHERE Sales_Order_Detail_ID = 10SSN

F16: Insert/update/delete/retrieve a Wholesaler

Create:

INSERT INTO `bike_store_final`.`Wholesaler` (`Wholesaler_ID`, `Wholesaler_Name`, `Contact`, `Phone`, `Email`) VALUES ('4', 'JKL', 'Jerry', '9974562355', 'Jerry@jkl.com');

Read:

SELECT *
FROM Wholesaler

Update:

UPDATE Wholesaler
SET Contact = "Jamie"
WHERE Wholesaler_ID = 4= '4984564512'

Delete:

DELETE FROM Wholesaler
WHERE Wholesaler_ID = 4

Appendix B - Data Dictionary Index

Column Name	Table Name
Address	Customer
Category_ID	Product
Category_ID	Product_Category
Category_Name	Product_Category
City	Customer
Contact	Wholesaler
Current_Balance	Customer
Customer_ID	Customer
Customer_ID	Customer_Payment
Customer_ID	Sales_Order

Customer_Payment_ID	Customer_Payment
Customer_Payment_ID	Sales_Order
Department_ID	Customer_Payment
Department_ID	Department
Department_ID	Employee
Department_Name	Customer_Payment
Department_Name	Department
Dependent_ID	Dependent
DOB	Employee
Email	Wholesaler
Employee_Dependent_Relation_ID	Dependent
Employee_Dependent_Relation_ID	Employee_Dependent_Relation
Employee_Dependent_Relation_Name	Employee_Dependent_Relation
Employee_ID	Dependent
Employee_ID	Employee
Employee_ID	Employee_Company_Project
Employee_ID	Sales_Order
First_Name	Customer
First_Name	Dependent
First_Name	Employee
Hire_Date	Employee
Job_Title_ID	Employee
Job_Title_ID	Job_Title
Job_Title_Name	Job_Title
Last_Name	Customer
Last_Name	Dependent
Last_Name	Employee
Payment_Type_ID	Customer_Payment
Payment_Type_ID	Payment_Type
Payment_Type_Name	Payment_Type
Phone	Wholesaler
Price	Sales_Order_Detail
Primary_Phone	Employee
Product_ID	Product
Product_ID	Product_Wholesaler
Product_ID	Sales_Order_Detail
Product_Name	Product
Product_Price	Product
Project_ID	Company_Project

Project_ID	Employee_Company_Project
Project_Name	Company_Project
Sales_Order_Date	Sales_Order
Sales_Order_Detail_ID	Sales_Order_Detail
Sales_Order_ID	Sales_Order
Sales_Order_ID	Sales_Order_Detail
State	Customer
Supervisor_Employee_ID	Employee
Units	Sales_Order_Detail
Units_In_Stock	Product
Wholesaler_ID	Product_Wholesaler
Wholesaler_ID	Wholesaler
Wholesaler_Name	Wholesaler
WSP	Product_Wholesaler
Zip	Customer

References

1. Fundamentals of Database Systems (7th Edition), Elmasri and Navathe
2. https://dev.mysql.com/doc/refman/8.0/en/