Technical Report

BUAN 6320 - Database Foundations for Analytics

Professor: Gasan Elkhodari

Group 11 Members:

Joshita Sairupa Allu
Priyank Chilamkuri
Jingyi Guo
Saloni Sandeep Juwatkar
Bryan Lee

UTD - JSOM

BUAN 6320 Database Foundations for Business Analytics
Project 1 Technical Report

INTRODUCTION

This Database definition document explains the design and implementation of the database that has been created to store data of various constituents involved in a Retail Store. The database will enable the creation and maintenance of retail stores which will increase the efficiency of the store management.

PURPOSE

The purpose of Retail Database Management System is to amplify the efficiency, sales and quality of the Retail Store services. It will simplify the process of maintaining data related to the inventory trends, orders, customers, employee management and finances. This system will not only help to overcome the manual errors but also keeps the store organized for better customer service.

PROJECT SCOPE

The Retail management database will help to create a system to improve the sales of the retail store. In-scope work will include documenting the project requirements, modeling the database in entity-relationship form, writing data definition language (DDL) SQL scripts to define and implement the database, and writing example data manipulation language (DML) and Structured Query Language (SQL) scripts to demonstrated the intended use of the database.

- In Scope Requirement:
    - Project requirements documentation
    - Entity-relationship model
    - DDL Scripts
    - Example DML Scripts
    - Example SQL Scripts
    - Comprehensive Report
    - Development of in-game level browser
    - Implementation of level browser server backend

Database Goals, Expectations, and Deliverables

Upon the completion of this project, the database shall contain fields for all level metadata, as well as a surrogate key for each level to avoid name collisions. Deliverables include this statement of work document, an entity-relationship diagram defining the structure of the database, DDL scripts for creating the database, example DML and SQL scripts that demonstrate proper usage of the database, and a final report on the project as a whole.

SQL Usage and Style

Adapted from Simon Holywell's SQL style guide, available at http://sqlstyle.guide/.

General

Naming Conventions PROJECT #1 TECHNICAL REPORT 11

- Use consistent and descriptive identifiers and names.
- Use white space and indentation to make code easier to read.
- Store time and date formation in ISO-8601 format (YYYY-MM-DD HH:MM:SS.SSSSS).
- Avoid redundant SQL, such as unnecessary quoting or parentheses or WHERE clauses that can be derived.
- Use C-style comments with opening /* and closing /* digraphs whenever possible; otherwise, precede comments with -- and finish them with a new line.
- For the sake of quick readability, prefer snake_case over CamelCase.
- Avoid Hungarian notation and other descriptive prefixes.
- Favor collective nouns over plurals, such as using staff instead of employees.
- When using quoted identifiers, use SQL92 double quotes to preserve portability.
- Avoid applying object-oriented design principles to SQL or database structures.

Naming Conventions

- Ensure that all names are unique and do not conflict with reserved keywords.
- Keep name length to 30 bytes; this usually means 30 characters, unless the name uses a multi-byte character set.
- Names must begin with a letter and may not end with an underscore.
- Names may contain only letters, numbers, and underscores.
- Multiple consecutive underscores are not allowed.
- Use underscores to represent spaces in names, e.g. “first name” becomes first_name.
- Avoid abbreviations; if it is necessary to use them, ensure they are commonly known and understood.
- Prefer collective nouns for table names.
- Tables and columns should never share the same name.
- Avoid concatenating the names of two tables when naming their relationship table.
- When naming columns, always prefer singular nouns.
- Avoid the name id for primary keys.
- Use lowercase in column names whenever reasonable.
- Use commonly-known suffixes to indicate the purpose of a column: _id, _name, _size, _addr, etc.

Query Syntax

- Always use uppercase for reserved keywords such as SELECT or WHERE.
- Prefer full-length keywords over abbreviated forms.
- Avoid database management system-specific keywords when an ANSI SQL equivalent already exists.
- Do not remove natural language spaces.

- Spaces should be used to line up code so that the root keywords all end on the same character boundary. This will improve the ability to scan the code quickly.
- Always include spaces around equals signs (=), after commas (,), and surrounding apostrophes (‘) where not within parentheses or with a trailing comma or semicolon.
- Always include newlines before AND or OR, after semicolons, after keyword definitions, and to separate code into related sections.
- Joins should be indented so that they line up with each other.
- Subqueries should be indented to the right and then laid out using the same style as a normal query.
- Where possible, use BETWEEN instead of connecting multiple statements with AND.
- Where possible, use IN()instead of multiple OR clauses.
- Use the CASE expression to interpret values before leaving the database.
- Avoid the use of UNION clauses and temporary tables whenever possible.

CREATE SYNTAX

- Avoid vendor-specific data types whenever possible, as they are not portable.
- Prefer the NUMERIC and DECIMAL types over the REAL and FLOAT types, save for situations in which floating-point math are strictly necessary.
- Default values must always be of the same type as their column.
- Default values must follow the data type declaration and come before any NOT NULL statement.
- Keys should be unique to some degree.
- Keys should be chosen from those columns whose data types are less likely to change in the future.
- Keys should only hold values that can be validated against a standardized format.

- Keys should be kept as simple as possible, but compound keys should still be used where necessary.
- To be complete and useful, tables must have at least one key.
- Constraints other than UNIQUE, PRIMARY KEY, and FOREIGN KEY should be given descriptive custom names.
- Consider placing multi-column constraints as close to both column definitions as possible; in more difficult cases, include them at the end of the CREATE TABLE definition.
- Table-level constraints that apply to an entire table should also appear at the end.
- Use alphabetical order where ON DELETE comes before ON UPDATE.
- If it makes sense to do so, align each part of a query at the same character position in each line, so as to help preserve readability.
- Use LIKE and SIMILAR TO constraints to ensure the integrity of strings whose format is known.
- If the range of a numerical value is known, CHECK() should be used to avoid incorrect values or the silent truncation of data.
- Avoid separating values and their units into their own columns; the column should make its values' units self-evident.
- Specialist products should be used to handle schema-less data such as Entity Attribute Value (EAV) tables.
- Avoid splitting data across multiple tables to satisfy external concerns, such as physical location or time-based archiving.

Project Management Methodology

The initial design of the database may be carried out in a linear fashion similar to simple interpretations of the waterfall model. This early implementation should seek to satisfy the preliminary database requirements established at the outset of the parent project. Following the completion of that implementation, the database team should switch to a project management methodology that emphasizes rapid iteration; ideally this should be the same methodology the software development team is using, to help facilitate cooperation and communication between the two teams. From that point on, the database team should revise the database design iteratively based on changes made to the software project's design and on feedback from the software development team.

Business Rules - Cardinality and Directions

1. A STORE may have many EMPLOYEES
2. An EMPLOYEE may work at only one STORE
3. A STORE may have many PRODUCTS
4. A PRODUCT may be sold at only one STORE
5. A STORE may have many INVOICES
6. An INVOICE may belong to only one STORE
7. An EMPLOYEE may deal with many INVOICES
8. An INVOICE may be dealt with by only one EMPLOYEE
9. A PRODUCT may be put on many INVOICES
10. An INVOICE may only have one PRODUCT
11. An INVOICE may only refer to one CUSTOMER
12. A CUSTOMER may have many different INVOICES

Functionalities

These functionalities we have developed in the Retail Store Management System.

- Manage Product Items
    - Adding New Product Items
    - Edit the Existing Product Items
    - View details of the Product Items
    - Listing of all Product Items

- Manage Sales
    - Adding New Sales

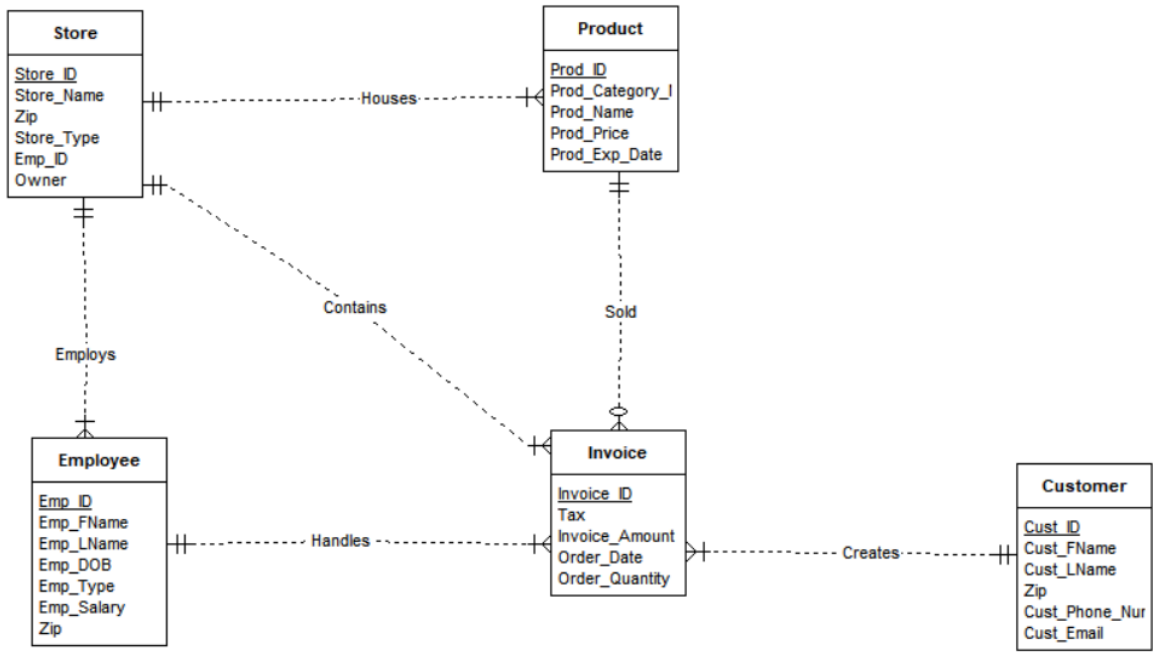
- Edit the Existing Sales
- View details of the Sales
- Listing of all Sales

- Reports of the project Retail Store Management System
  - Report of all Customers
  - Report of all Employees
  - Report of all Product Items
  - Report of all Sales

ASSUMPTIONS

- While employees could potentially fail to resolve an invoice, we will consider the final involved employee as the resolver to count towards employee consideration under invoice.
- While employees may move and work under multiple stores, we will consider them the employee under the main store that gives them their salary as to avoid cluttering
- 'Customer' refers to the individual paying for the Product and not any other auxiliary individuals with them
- 'Product' refers to a singular item and not the entire product line or type that it falls under

ERD DIAGRAM



Entity and Attribute Description

Store (Entity) - Profile information for the 'Store' object referring to places that items can be sold such as supermarkets or smaller individually owned stalls
Store_ID (Primary Key) - The unique ID of the store
Emp_ID (Foreign Key) - The ID for the employee working at the store inherited from the employee entity
ZIP - The postal zip for the store
Store_Name - The name of the store
Owner - The name of the owner of the store
Store_Type - The type of the store

Employee (Entity) - Profile information for the employees working at the store
Employee.Emp_ID (Primary Key) - The unique ID of the employee
Store_ID (Foriegn Key) - The unique ID for the store inherited from store entity
Zip - The postal zip for employees
Emp_DOB - the date of birth of the employee
Emp_FName - The first name of the employee
Emp_LName - The last name of the employee
Emp_Salary - The salary of the employee
Emp_Type - The employee type of the employee

Product (Entity) - The information on the products being sold in the store
Prod_ID (Primary Key) - The unique ID identifier of each product
Store_ID (Foreign Key) - The ID of the store the product is being sold at inherited from the store entity
Prod_Category_Name - The category name of the products that helps to organize them
Prod_Exp_Date - The expiration date of the product
Prod_Name - The name of the product
Prod_Price - The price of the product

Invoice (Entity) - Information in the invoices being generated whenever a customer purchases a product from the store
Invoice_ID - The unique ID of the invoice being generated
Prod_ID (Foreign Key) - The ID of the product being generated from the invoice inherited from the product entity

Store_ID (Foreign Key) - The ID of the store the invoice is generated at inherited from the store entity
Cust_ID (Foreign Key) - The ID of the customer the invoice is from inherited from the customer entity
Emp_ID (Foreign Key) - The ID of the employee that is handling the invoice inherited from the employee entity
Invoice_Amount - The amount to be paid on the invoice
Order_Date - The date the invoice was made
Order_Quantity - The quantity of products on the invoice
Invoice_Tax - The tax rate on the invoice

Customer (Entity) - The information on the customer
Cust_ID (Primary Key) - The unique ID of the customer
Zip - The postal zip of the customer
Cust_Email - the email of the customer
Cust_FName - The first name of the customer
Cust_LName - The last name of the customer
Cust_Phone_Num - The phone number of the customer

Relationship and Cardinality Description.

Relationship: Houses between STORE and PRODUCT
Cardinality: 1:M between STORE and PRODUCT
Business Rule: A STORE may contain one or many PRODUCT; a PRODUCT may be sold at only one store

Relationship: Employs between STORE and EMPLOYEE
Cardinality: 1:M between STORE and EMPLOYEE
Business Rule: A STORE may employ one or many EMPLOYEE; an EMPLOYEE may work at only one store

Relationship: Handles between EMPLOYEE and INVOICE
Cardinality: 1:M between EMPLOYEE and INVOICE
Business Rule: An EMPLOYEE may handle one or many INVOICE; an INVOICE may be handled by only one EMPLOYEE

Relationship: Sold between PRODUCT and INVOICE
Cardinality: 1:M between PRODUCT and INVOICE
Business Rule: A PRODUCT may be sold on zero or many INVOICES; An INVOICE may contain only one PRODUCT

Relationship: Contains between STORE and INVOICE
Cardinality: 1:M between STORE and INVOICE
Business Rule: A STORE may contain one or many INVOICE; an INVOICE may be held by only one STORE

Relationship: Creates between CUSTOMER and INVOICE
Cardinality: 1:M between CUSTOMER and INVOICE
Business Rule: A CUSTOMER may create one or many INVOICE; an INVOICE may be created by only one CUSTOMER

DDL Source Code

```
/*
Project - DDL Project BUAN 6320
*/


/* DROP statements to clean up objects from previous run */
-- Triggers
DROP TRIGGER TRG_Stores;
DROP TRIGGER TRG_Invoices;
DROP TRIGGER TRG_Customers;

-- Sequences
DROP SEQUENCE SEQ_Invoices_Invoice_ID;
DROP SEQUENCE SEQ_Stores_Store_ID;
DROP SEQUENCE SEQ_Customers_Cust_ID;

-- Indices
DROP INDEX IDX_Stores_Name;

DROP INDEX IDX_Customers_Cust_Email;
DROP INDEX IDX_Customers_Cust_Phone_Num;

DROP INDEX IDX_Invoices_Cust_ID_FK;
DROP INDEX IDX_Invoices_Store_ID_FK;
DROP INDEX IDX_Invoices_Emp_ID_FK;
DROP INDEX IDX_Invoices_Prod_ID_FK;

-- Tables
DROP TABLE Invoices;
DROP TABLE Customers;
DROP TABLE Products;
DROP TABLE Employees;
DROP TABLE Stores;
```

```
/* Create tables based on entities */
CREATE TABLE Stores
(
Store_ID int,
Store_Name varchar(255) NOT NULL,
Store_Type varchar(25) NOT NULL,
Emp_ID int,
Owners varchar(255) NOT NULL,
Zip int,
CONSTRAINT PK_Stores PRIMARY KEY (Store_ID)
);

CREATE TABLE Products
(
Prod_ID int,
Prod_Category_ID int,
Store_ID int,
Prod_Name varchar(225) NOT NULL,
Prod_Price varchar(25) NOT NULL,
Prod_Exp_Date varchar(25) NOT NULL,
CONSTRAINT PK_Products PRIMARY KEY (Prod_ID),
CONSTRAINT FK_Products FOREIGN KEY (Store_ID) REFERENCES Stores (Store_ID)
);


CREATE TABLE Employees
(
Emp_ID int,
Store_ID int,
Emp_FName varchar(225) NOT NULL,
Emp_LName varchar(225) NOT NULL,
Emp_DOB varchar(25) NOT NULL,
Emp_Type varchar(25) NOT NULL,
Emp_Salary int,
Zip int,
CONSTRAINT PK_Employees PRIMARY KEY (Emp_ID),
CONSTRAINT FK_Employees FOREIGN KEY (Store_ID) REFERENCES Stores (Store_ID)
);

CREATE TABLE Customers
(
Cust_ID int,
Cust_FName varchar(225) NOT NULL,
Cust_LName varchar(225) NOT NULL,
Cust_Phone_Num varchar(25) NOT NULL,
Cust_Email varchar(25) NOT NULL,
Zip int,
```

```
CONSTRAINT PK_Customers PRIMARY KEY (Cust_ID)
);

CREATE TABLE invoices
(
Invoice_ID int,
Cust_ID int,
Store_ID int,
Emp_ID int,
Taxable varchar(5) NOT NULL,
Prod_ID int,
Invoice_Amount varchar(225) NOT NULL,
OQuantity int,
Order_Date varchar(225) NOT NULL,
CONSTRAINT PK_Invoices PRIMARY KEY (Invoice_ID),
CONSTRAINT FK_Invoices FOREIGN KEY (Store_ID) REFERENCES Stores (Store_ID),
CONSTRAINT FK_Invoices_Cust_ID FOREIGN KEY (Cust_ID) REFERENCES Customers
(Cust_ID),
CONSTRAINT FK_Invoices_Emp_ID FOREIGN KEY (Emp_ID) REFERENCES Employees
(Emp_ID),
CONSTRAINT FK_Invoices_Prod_ID FOREIGN KEY (Prod_ID) REFERENCES Products
(Prod_ID)
);

/* Create indices for natural keys, foreign keys, and frequently-queried
columns */
-- Stores
-- Natural Keys
CREATE INDEX IDX_Stores_Name ON Stores (Store_Name);

-- Customers
-- Natural Keys
CREATE INDEX IDX_Customers_Cust_Email ON Customers (Cust_Email);
CREATE INDEX IDX_Customers_Cust_Phone_Num ON Customers (Cust_Phone_Num);

-- Invoices
-- Foreign Keys
CREATE INDEX IDX_Invoices_Cust_ID_FK ON Invoices (Cust_ID);
CREATE INDEX IDX_Invoices_Store_ID_FK ON Invoices (Store_ID);
CREATE INDEX IDX_Invoices_Emp_ID_FK ON Invoices (Emp_ID);
CREATE INDEX IDX_Invoices_Prod_ID_FK ON Invoices (Prod_ID);

/* Alter Tables by adding Audit Columns */
ALTER TABLE Stores
ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
```

```
);

ALTER TABLE Products ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

ALTER TABLE Employees ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

ALTER TABLE Customers ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

ALTER TABLE Invoices ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

/* Create Sequences */
CREATE SEQUENCE SEQ_Invoices_Invoice_id
INCREMENT BY 1
START WITH 81000000
NOMAXVALUE
MINVALUE 81000000
NOCACHE;

CREATE SEQUENCE SEQ_Stores_Store_ID
INCREMENT BY 1
START WITH 1
NOMAXVALUE
MINVALUE 1
NOCACHE;

CREATE SEQUENCE SEQ_Customers_Cust_ID
INCREMENT BY 1
START WITH 100
NOMAXVALUE
```

```
MINVALUE 100
NOCACHE;

/* Create Triggers */
-- Business purpose: The TRG_Stores trigger automatically assigns a
sequential user ID to a newly-inserted row in the Stores table, as well as
setting the join date to the current system date and assigning appropriate
values to the created_by and date_created fields. If the record is being
inserted or updated, appropriate values are assigned to the modified_by and
modified_date fields.
CREATE OR REPLACE TRIGGER TRG_Stores
    BEFORE INSERT OR UPDATE ON Stores
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.Store_ID IS NULL THEN
                :NEW.Store_ID := SEQ_Stores_Store_ID.NEXTVAL;
            END IF;
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
        End IF;
END;
/

-- Business purpose: The TRG_Invoices trigger automatically assigns as
sequential invoice_ID to a newly-inserted row in the invocies table, as well
as setting the upload date to the current system date, as well as setting the
join date to the current system date and assigning appropriate values to the
created_by and date_created fields. If the record is being inserted or
updated, appropriate values are assigned to the modified_by and modified_date
fields.
CREATE OR REPLACE TRIGGER TRG_Invoices
    BEFORE INSERT OR UPDATE ON Invoices
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.Invoice_ID IS NULL THEN
                :NEW.Invoice_ID := SEQ_Invoices_Invoice_ID.NEXTVAL;
            END IF;
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
```

```
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Business purpose: The TRG_Customers trigger sets the modified_by and
date_modified fields to appropriate values in a newly inserted or updated
record; if the record is being inserted, then the created_by and date_created
fields are set to appropriate values too.
CREATE OR REPLACE TRIGGER TRG_Customers
    BEFORE INSERT OR UPDATE ON Customers
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.Cust_ID IS NULL THEN
                :NEW.Cust_ID := SEQ_Customers_Cust_ID.NEXTVAL;
            END IF;
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Business purpose: The TRG_Products trigger sets the modified_by and
date_modified fields to appropriate values in a newly inserted or updated
record; if the record is being inserted, then the created_by and date_created
fields are set to appropriate values too.
CREATE OR REPLACE TRIGGER TRG_Products
    BEFORE INSERT OR UPDATE ON Products
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
```

```
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Business purpose: The TRG_Employees trigger sets the modified_by and
date_modified fields to appropriate values in a newly inserted or updated
record; if the record is being inserted, then the created_by and date_created
fields are set to appropriate values too.
CREATE OR REPLACE TRIGGER TRG_Employees
    BEFORE INSERT OR UPDATE ON Employees
    FOR EACH ROW
    BEGIN
        IF INSERTING THEN
            IF :NEW.created_by IS NULL THEN
                :NEW.created_by := USER;
            END IF;
            IF :NEW.date_created IS NULL THEN
                :NEW.date_created := SYSDATE;
            END IF;
        END IF;
        IF INSERTING OR UPDATING THEN
            :NEW.modified_by := USER;
            :NEW.date_modified := SYSDATE;
        END IF;
END;
/

-- Check the DBMS data dictionary to make sure that all objects have been
created successfully
SELECT TABLE_NAME FROM USER_TABLES;

SELECT OBJECT_NAME, STATUS, CREATED, LAST_DDL_TIME FROM USER_OBJECTS;
```


DML Statements

```
/* Populate all tables */
-- Customers Table
Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Bruce', 'Smith', '469-999-1123', 'bruce.smith@outlook.com',
'75039');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
```

```
VALUES ('Andrew', 'Miller', '469-222-1123', 'andrew.miller@gmail.com',
'75039');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Angela', 'Nelson', '469-533-1123', 'angela.nelson@hotmai.com',
'75039');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Chris', 'Lee', '469-433-0909', 'chris.lee@163.com', '75039');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Christine', 'Bernstein', '469-091-0909', 'christine.b@hotmail.com',
'75080');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Sam', 'Cooper', '469-087-0733', 'sam.cooper@gmail.com', '75023');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Carmen', 'Powell', '469-987-5434', 'carmen.powell@outlook.com',
'75001');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Karen', 'Trump', '469-091-3375', 'karen.trump@hotmail.com',
'75022');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Brad', 'Tayler', '469-992-7734', 'brad.tayler@gmail.com', '75023');

Insert into Customers (Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email,
Zip)
VALUES ('Bratt', 'Swift', '469-371-8780', 'bratt.swift@gmail.com', '75002');

-- Stores Table
INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Allen', 'Supermarket',
'96033', 'Peter Kim', '75002');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Dallas', 'Supermarket',
'96028', 'Mary Morels', '75257');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
```

```
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Plano', 'Supermarket',
'96001', 'Chaz Mathew', '75023');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Houston', 'Supermarket',
'96077', 'Brian Quinn', '77005');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Austin', 'Supermarket',
'96035', 'Michael Martinize', '73301');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Rockwall', 'Neighborhood',
'96022', 'Nancy Abraham', '75189');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Irving', 'Neighborhood',
'96099', 'David Jonathan', '75039');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-San Antonio', 'Supermarket',
'96082', 'Tony Green', '78112');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Prosper', 'Neighborhood',
'96047', 'Emma Arroyo', '75033');

INSERT into Stores (store_id,Store_Name, Store_Type, Emp_ID, Owners, Zip)
VALUES (SEQ_Stores_Store_ID.nextval,'Little Red-Sherman', 'Neighborhood',
'96021', 'Sherry Hugh', '75090');

-- Employees Table
Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('550001', '1', 'Tiffany', 'Kimberly', '01/02/1990', 'Full Time',
'50000.00', '75080');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('550712', '9', 'Jason', 'Smith', '02/07/1992', 'Full Time',
'60000.00', '75237');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('550456', '9', 'Micah', 'Young', '05/12/19790', 'Full Time',
'70000.00', '75080');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
```

```
VALUES ('540074', '1',  'Sherry', 'Hill', '04/02/1983', 'Part Time', '13',
'75023');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('540002', '9', 'Donnie', 'Lankford', '11/02/1998', 'Part Time', '35',
'77005');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('540078', '9', 'Lauren', 'Huang', '12/02/1989', 'Part Time', '31',
'75189');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('540098', '1', 'Eunice', 'Sue', '09/30/1995', 'Part Time', '18',
'75189');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('540101', '6', 'Kai', 'Zhang', '08/17/1996', 'Part Time', '50',
'75189');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('550234', '2', 'Sonia', 'Abraham', '01/02/1990', 'Full Time',
'90000.00', '75023');

Insert into Employees (Emp_ID, Store_ID, Emp_FName, Emp_LName, Emp_DOB,
Emp_Type, Emp_Salary, Zip)
VALUES ('550651', '4', 'Keith', 'Clein', '12/24/1997', 'Full Time',
'130000.00', '75023');

-- Products Table
INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('320001', '320000', '2', 'CIRCUIT,BREAKER,80A,1P,C CURVE,240VAC',
'206.73', '05/14/2037');

INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('320006', '320000', '3', 'GAUGE,PRESSURE,700LFB4002LA140,300PSI',
'12.34','07/20/2033');

INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('310002', '310000', '4', 'ANGLE,A36,L2"x2"x1/4"20DOMESTIC', '5.13',
'01/04/2023');
```

```
INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('310005', '310000', '5', 'CHANNEL,A36,C4x5.420DOMESTIC', '0.76',
'01/04/2023');

INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('330008', '310000', '6', 'MUNICIPAL UTILITY JOB', '30152.00',
'12/31/2023');

INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('330897', '310000', '1', 'FIRE HOUSE', '60808.00', '12/31/2023');

INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('320769', '320000', '1', 'PUMP,CDF32-2/2-D0HD2B', '1387.23',
'07/20/2033');

INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('310243', '310000', '10', 'TUBE,A500,2"x2"x1/4"20DOMESTIC', '3.14',
'01/04/2023');

INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('320157', '320000', '1', 'EXPANSION,TANK,ARMSTRONG,26', '4241.33',
'07/20/2033');

INSERT into Products (Prod_ID, Prod_Category_ID, Store_ID, Prod_Name,
Prod_Price, Prod_Exp_Date)
VALUES ('320154', '320000', '8', 'VALVE,TRIPLE,DUTY,AW,TDV-006GR,6"',
'3917.23', '07/20/2033');

-- Invoices Table
Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'101', '2', '550001', 'YES',
'320001', '671.36', '3', '01/12/2022');

Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'104', '6', '550456', 'YES',
'320001', '223.79', '1', '03/01/2022');

Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'105', '2', '550456', 'NO', '320001',
'2067.30', '10', '11/01/2022');
```

```
Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'106', '1', '550001', 'NO', '320154',
'3917.23', '1', '03/23/2022');

Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'107', '5', '540074', 'YES',
'320154', '4240.40', '1', '04/28/2022');

Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'109', '2', '550001', 'NO', '310243',
'3140', '100', '09/06/2022');

Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'100', '4', '540074', 'NO', '320157',
'4241.33','1', '12/01/2021');

Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'106', '6', '550001', 'YES',
'320157', '32138.68', '7', '07/01/2021');

Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'103', '4', '550456', 'YES',
'320157', '4591.24', '1', '02/01/2022');

Insert into Invoices (Invoice_ID,Cust_ID, Store_ID, Emp_ID, Taxable, Prod_ID,
Invoice_Amount, OQuantity, Order_Date)
VALUES (SEQ_Invoices_Invoice_id.nextval,'108', '8', '550712', 'NO', '310243',
'3.14', '1', '01/02/2022');


-- Queries
--query 1
SELECT *
FROM Employees;

--query 2
SELECT Cust_ID, Cust_FName, Cust_LName, Cust_Phone_Num, Cust_Email
FROM Customers;

--query 3
SELECT *
FROM Stores.zip;
```

```
--query 4
SELECT *
FROM Customers INNER JOIN invoices
ON Customers.Cust_ID=invoices.Cust_ID;

--query 5
SELECT *
FROM invoices
ORDER BY Invoice_Amount;

--query 6
SELECT Invoice_ID, Store_Name, OQuantity, Prod_Name, Prod_Price
FROM invoices JOIN Products on invoices.Prod_ID=Products.Prod_ID JOIN Stores
on Products.Store_ID=Stores.Store_ID
FETCH FIRST 10 ROWS ONLY;

--query 7
SELECT DISTINCT Products.*, Stores.*, Invoices.*
FROM Products JOIN Stores on Products.Store_ID=Stores.Store_ID JOIN Invoices
on Products.Prod_ID=Invoices.Prod_ID;

--query 8
SELECT Store_ID
FROM Products
GROUP BY Store_ID
HAVING Store_ID>'2';

--query 9
SELECT *
FROM Stores
WHERE Emp_ID IN ('96033', '96077', '96099', '96021');

--query 10
SELECT LENGTH(Emp_ID)
FROM Employees;

--query 11
SELECT *
FROM Customers;


DELETE
FROM Customers
WHERE Cust_FName IN ('Bruce','Andrew');

SELECT *
FROM Customers;
ROLLBACK;
```

```
--query 12
SELECT *
FROM invoices;

UPDATE invoices
SET Invoice_Amount='0'
WHERE Taxable='NO';

SELECT *
FROM invoices

--Advanced Queries
--Q.13 list number of customers in each zip code, list from High to low.
SELECT COUNT(Cust_ID), Zip
FROM Customers
GROUP BY Zip
ORDER BY COUNT(Cust_ID) DESC;

--Q14 List all the customer who haven't make any purchase in 2022
SELECT Cust_FName,Cust_LName
FROM customers
WHERE Cust_ID NOT IN (SELECT Cust_ID FROM Invoices WHERE Order_date >
'01/01/2022')
ORDER BY Cust_FName, Cust_LName

--Q.15 calculate the revenue for every store and rank
SELECT  stores.store_name,
        stores.store_ID,
        SUM(invoices.Invoice_Amount) AS revenue,
        RANK() OVER (ORDER BY SUM(invoices.Invoice_Amount) DESC) AS
revenue_rank
FROM invoices inner join stores on stores.store_ID = invoices.store_ID
GROUP BY stores.store_name, stores.store_ID

--Q.16 calculate the difference between each month's revenue and the previous
month for the purchase made after June,1 2022
SELECT  Order_date,
        Invoice_Amount,
        Invoice_Amount - LAG(Invoice_Amount, 1) OVER (ORDER BY Order_date) AS
monthly_delta
FROM Invoices
WHERE Order_date > '06/01/2022'
ORDER BY Order_date, Invoice_Amount

--Q.17 Display the product and the product name of all products which have
more than one sales record.
SELECT p.Prod_ID, p.Prod_Name
FROM (
```

```
      SELECT Products.Prod_ID, Products.Prod_Name
      FROM Products INNER JOIN Invoices ON Products.Prod_ID =
Invoices.Prod_ID
      ) p
GROUP BY p.Prod_ID, p.Prod_Name
HAVING COUNT(p.Prod_ID) > 1;

--Q.18 Find the customer who purchased the most often and show their ID and
the amount of that purchase.
SELECT c1.Cust_ID, c4.Invoice_Amount
FROM(
      SELECT Invoices.Cust_ID, COUNT(Invoices.Cust_ID) AS num_purchased
      FROM Invoices INNER JOIN Customers ON Invoices.Cust_ID =
      Customers.Cust_ID
      GROUP BY Invoices.Cust_ID
      ) c1,
      (SELECT MAX(c2.num_purchased) AS max_purchased
            FROM (
                  SELECT Invoices.Cust_ID, COUNT(Invoices.Cust_ID) AS
num_purchased
                  FROM Invoices INNER JOIN Customers ON Invoices.Cust_ID =
                  Customers.Cust_ID
            GROUP BY Invoices.Cust_ID
                  ) c2
      ) c3,
      (SELECT Customers.Cust_FName, Customers.Cust_LName,
Customers.Cust_ID,Invoices.Invoice_Amount
            From Customers INNER JOIN Invoices on Invoices.Cust_ID =
            Customers.Cust_ID) c4
where c1.num_purchased = c3.max_purchased
and c1.Cust_ID = c4.Cust_ID

--Q19query to display customer id ,first name, last name, invoice, order
quantity who has ordered more than one quantity
SELECT c.cust_id,Cust_FName,Cust_LName,
i.OQuantity
FROM customers c
JOIN invoices i on
c.cust_id = i.cust_id
group by c.cust_id,Cust_FName,Cust_LName,
i.OQuantity
having i.OQuantity>1;

--Q20 Display Sum on Invoice amount for the Invoices created after
'04/01/2022' for customer residing in 75039 zip code.
SELECT Order_date,sum(i.invoice_amount)
    FROM Invoices i
    where i.cust_id in
        (select c.cust_id
```

```
            from customers c
            where c.zip = '75039')
    group by i.invoice_amount,Order_date having Order_date > '04/01/2022';
```

DDL, DDL Outputs

```
Trigger TRG_STORES dropped.

Trigger TRG_INVOICES dropped.

Trigger TRG_CUSTOMERS dropped.

Sequence SEQ_INVOICES_INVOICE_ID dropped.

Sequence SEQ_STORES_STORE_ID dropped.

Sequence SEQ_CUSTOMERS_CUST_ID dropped.

Index IDX_STORES_NAME dropped.

Index IDX_CUSTOMERS_CUST_EMAIL dropped.

Index IDX_CUSTOMERS_CUST_PHONE_NUM dropped.

Index IDX_INVOICES_CUST_ID_FK dropped.

Index IDX_INVOICES_STORE_ID_FK dropped.

Index IDX_INVOICES_EMP_ID_FK dropped.

Index IDX_INVOICES_PROD_ID_FK dropped.
```

```
Table INVOICES dropped.

Table CUSTOMERS dropped.

Table PRODUCTS dropped.

Table EMPLOYEES dropped.

Table STORES dropped.

Table STORES created.

Table PRODUCTS created.

Table EMPLOYEES created.

Table CUSTOMERS created.

Table INVOICES created.

Index IDX_STORES_NAME created.

Index IDX_CUSTOMERS_CUST_EMAIL created.

Index IDX_CUSTOMERS_CUST_PHONE_NUM created.

Index IDX_INVOICES_CUST_ID_FK created.

Index IDX_INVOICES_STORE_ID_FK created.

Index IDX_INVOICES_EMP_ID_FK created.
```

```
Index IDX_INVOICES_PROD_ID_FK created.


Table STORES altered.


Table PRODUCTS altered.


Table EMPLOYEES altered.


Table CUSTOMERS altered.


Table INVOICES altered.


Sequence SEQ_INVOICES_INVOICE_ID created.


Sequence SEQ_STORES_STORE_ID created.


Sequence SEQ_CUSTOMERS_CUST_ID created.


Trigger TRG_STORES compiled


Trigger TRG_INVOICES compiled


Trigger TRG_CUSTOMERS compiled


Trigger TRG_PRODUCTS compiled


Trigger TRG_EMPLOYEES compiled

>>Query Run In:Query Result 14
>>Query Run In:Query Result 15

1 row inserted.


1 row inserted.
```

```
1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.
```

```
1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.
```

```
1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.

1 row inserted.
```


QUERY OUTPUT:

Query 1:

	EMP_ID	STORE_ID	EMP_FNAME	EMP_LNAME	EMP_DOB	EMP_TYPE	EMP_SALARY	ZIP	CREATED_BY	DATE_CREATED	MODIFIED_BY	DATE_MODIFIED
1	550001	1	Tiffany	Kimberly	01/02/1990	Full Time	50000	75080	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
2	550712	9	Jason	Smith	02/07/1992	Full Time	60000	75237	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
3	550456	9	Micah	Young	05/12/19790	Full Time	70000	75080	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
4	540074	1	Sherry	Hill	04/02/1983	Part Time	13	75023	STUDENT_D	14-NOV-22	STUDENT_DB1	14-NOV-22
5	540002	9	Donnie	Lankford	11/02/1998	Part Time	35	77005	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
6	540078	9	Lauren	Huang	12/02/1989	Part Time	31	75189	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
7	540098	1	Eunice	Sue	09/30/1995	Part Time	18	75189	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
8	540101	6	Kai	Zhang	08/17/1996	Part Time	50	75189	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
9	550234	2	Sonia	Abraham	01/02/1990	Full Time	90000	75023	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
10	550651	4	Keith	Clein	12/24/1997	Full Time	130000	75023	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22

Query 2

	CUST_ID	CUST_FNAME	CUST_LNAME	CUST_PHONE_NUM	CUST_EMAIL
1	100	Bruce	Smith	469-999-1123	bruce.smith@outlook.com
2	101	Andrew	Miller	469-222-1123	andrew.miller@gmail.com
3	102	Angela	Nelson	469-533-1123	angela.nelson@hotmai.com
4	103	Chris	Lee	469-433-0909	chris.lee@163.com
5	104	Christine	Bernstein	469-091-0909	christine.b@hotmail.com
6	105	Sam	Cooper	469-087-0733	sam.cooper@gmail.com
7	106	Carmen	Powell	469-987-5434	carmen.powell@outlook.com
8	107	Karen	Trump	469-091-3375	karen.trump@hotmail.com
9	108	Brad	Tayler	469-992-7734	brad.tayler@gmail.com
10	109	Bratt	Swift	469-371-8780	bratt.swift@gmail.com

Query 4

	CUST_ID	CUST_FNAME	CUST_LNAME	CUST_PHONE_NUM	CUST_EMAIL	ZIP	CREATED_BY	DATE_CREATED	MODIFIED_BY	DATE_MODIFIED	INVOICE_ID	CUST_ID_1	STORE_ID	EMP_ID	TAXABLE	PROD_ID	INVOICE
1	100	Bruce	Smith	469-999-1123	bruce.smith@outlook.com	75039	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000006	100	4	540074	NO	320157	4241.3
2	101	Andrew	Miller	469-222-1123	andrew.miller@gmail.com	75039	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000000	101	2	550001	YES	320001	671.36
3	103	Chris	Lee	469-433-0909	chris.lee@163.com	75039	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000008	103	4	550456	YES	320157	4591.2
4	104	Christine	Bernstein	469-091-0909	christine.b@hotmail.com	75080	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000001	104	6	550456	YES	320001	223.79
5	105	Sam	Cooper	469-087-0733	sam.cooper@gmail.com	75023	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000002	105	2	550456	NO	320001	2067.3
6	106	Carmen	Powell	469-987-5434	carmen.powell@outlook.com	75001	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000003	106	1	550001	NO	320154	3917.2
7	106	Carmen	Powell	469-987-5434	carmen.powell@outlook.com	75001	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000007	106	6	550001	YES	320157	32138.
8	107	Karen	Trump	469-091-3375	karen.trump@hotmail.com	75022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000004	107	5	540074	YES	320154	4240.4
9	108	Brad	Tayler	469-992-7734	brad.tayler@gmail.com	75023	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000009	108	8	550712	NO	310243	3.14
10	109	Bratt	Swift	469-371-8780	bratt.swift@gmail.com	75002	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	81000005	109	2	550001	NO	310243	3140

Query 5

	INVOICE_ID	CUST_ID	STORE_ID	EMP_ID	TAXABLE	PROD_ID	INVOICE_AMOUNT	OQUANTITY	ORDER_DATE	CREATED_BY	DATE_CREATED	MODIFIED_BY	DATE_MODIFIED
1	81000002	105	2	550456	NO	320001	2067.30	10	11/01/2022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
2	81000001	104	6	550456	YES	320001	223.79	1	03/01/2022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
3	81000009	108	8	550712	NO	310243	3.14	1	01/02/2022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
4	81000005	109	2	550001	NO	310243	3140	100	09/06/2022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
5	81000007	106	6	550001	YES	320157	32138.68	7	07/01/2021	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
6	81000003	106	1	550001	NO	320154	3917.23	1	03/23/2022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
7	81000004	107	5	540074	YES	320154	4240.40	1	04/28/2022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
8	81000006	100	4	540074	NO	320157	4241.33	1	12/01/2021	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
9	81000008	103	4	550456	YES	320157	4591.24	1	02/01/2022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22
10	81000000	101	2	550001	YES	320001	671.36	3	01/12/2022	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22

Query 6

	INVOICE_ID	STORE_NAME	OQUANTITY	PROD_NAME	PROD_PRICE
1	81000006	Little Red-Allen	1	EXPANSION,TANK,ARMSTRONG,26	4241.33
2	81000007	Little Red-Allen	7	EXPANSION,TANK,ARMSTRONG,26	4241.33
3	81000008	Little Red-Allen	1	EXPANSION,TANK,ARMSTRONG,26	4241.33
4	81000000	Little Red-Dallas	3	CIRCUIT,BREAKER,80A,1P,C CURVE,240VAC	206.73
5	81000001	Little Red-Dallas	1	CIRCUIT,BREAKER,80A,1P,C CURVE,240VAC	206.73
6	81000002	Little Red-Dallas	10	CIRCUIT,BREAKER,80A,1P,C CURVE,240VAC	206.73
7	81000003	Little Red-San Antonio	1	VALVE,TRIPLE,DUTY,AW,TDV-006GR,6"	3917.23
8	81000004	Little Red-San Antonio	1	VALVE,TRIPLE,DUTY,AW,TDV-006GR,6"	3917.23
9	81000005	Little Red-Sherman	100	TUBE,A500,2"x2"x1/4"20DOMESTIC	3.14
10	81000009	Little Red-Sherman	1	TUBE,A500,2"x2"x1/4"20DOMESTIC	3.14

Query 7

	PROD_ID	PROD_CATEGORY_ID	STORE_ID	PROD_NAME	PROD_PRICE	PROD_EXP_DATE	CREATED_BY	DATE_CREATED	MODIFIED_BY	DATE_MODIFIED	STORE_ID_1	STORE_NAME	S
1	320157	320000	1	EXPANSION,TANK,ARMSTRONG,26	4241.33	07/20/2033	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	1	Little Red-Allen	Su
2	320157	320000	1	EXPANSION,TANK,ARMSTRONG,26	4241.33	07/20/2033	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	1	Little Red-Allen	Su
3	320157	320000	1	EXPANSION,TANK,ARMSTRONG,26	4241.33	07/20/2033	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	1	Little Red-Allen	Su
4	320001	320000	2	CIRCUIT,BREAKER,80A,1P,C CURVE,240VAC	206.73	05/14/2037	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	2	Little Red-Dallas	Su
5	320001	320000	2	CIRCUIT,BREAKER,80A,1P,C CURVE,240VAC	206.73	05/14/2037	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	2	Little Red-Dallas	Su
6	320001	320000	2	CIRCUIT,BREAKER,80A,1P,C CURVE,240VAC	206.73	05/14/2037	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	2	Little Red-Dallas	Su
7	320154	320000	8	VALVE,TRIPLE,DUTY,AW,TDV-006GR,6"	3917.23	07/20/2033	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	8	Little Red-San Antonio	Su
8	320154	320000	8	VALVE,TRIPLE,DUTY,AW,TDV-006GR,6"	3917.23	07/20/2033	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	8	Little Red-San Antonio	Su
9	310243	310000	10	TUBE,A500,2"x2"x1/4"20DOMESTIC	3.14	01/04/2023	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	10	Little Red-Sherman	Ne
10	310243	310000	10	TUBE,A500,2"x2"x1/4"20DOMESTIC	3.14	01/04/2023	STUDENT_DB1	14-NOV-22	STUDENT_DB1	14-NOV-22	10	Little Red-Sherman	Ne

Query 8

	STORE_ID
1	6
2	8
3	4
4	5
5	10
6	3

Query 9

	STORE_ID	STORE_NAME	STORE_TYPE	EMP_ID	OWNERS	ZIP	CREATED_BY	DATE_CREATED	MODIFIED_BY	DATE_MODIFIED
1	1	Little Red-Allen	Supermarket	96033	Peter Kim	75002	STUDENT_DB1	14-NOV-22	(null)	(null)
2	4	Little Red-Houston	Supermarket	96077	Brian Quinn	77005	STUDENT_DB1	14-NOV-22	(null)	(null)
3	7	Little Red-Irving	Neighborhood	96099	David Jonathan	75039	STUDENT_DB1	14-NOV-22	(null)	(null)
4	10	Little Red-Sherman	Neighborhood	96021	Sherry Hugh	75090	STUDENT_DB1	14-NOV-22	(null)	(null)

Query 10

	LENGTH(EMP_ID)
1	6
2	6
3	6
4	6
5	6
6	6
7	6
8	6
9	6
10	6

Query 13

	COUNT(CUST_ID)	ZIP
1	4	75039
2	2	75023
3	1	75080
4	1	75022
5	1	75002
6	1	75001

Query 14

	CUST_FNAME	CUST_LNAME
1	Angela	Nelson

Query 15

	STORE_NAME	STORE_ID	REVENUE	REVENUE_RANK
1	Little Red-Rockwall	6	32362.47	1
2	Little Red-Houston	4	8832.57	2
3	Little Red-Dallas	2	5878.66	3
4	Little Red-Austin	5	4240.4	4
5	Little Red-Allen	1	3917.23	5
6	Little Red-San Antonio	8	3.14	6

Query 16

	ORDER_DATE	INVOICE_AMOUNT	MONTHLY_DELTA
1	07/01/2021	32138.68	(null)
2	09/06/2022	3140	-28998.68
3	11/01/2022	2067.30	-1072.7
4	12/01/2021	4241.33	2174.03

Query 17

	PROD_ID	PROD_NAME
1	320154	VALVE,TRIPLE,DUTY,AW,TDV-006GR,6"
2	320157	EXPANSION,TANK,ARMSTRONG,26
3	320001	CIRCUIT,BREAKER,80A,1P,C CURVE,240VAC
4	310243	TUBE,A500,2"x2"x1/4"20DOMESTIC

Query 18

	CUST_ID	INVOICE_AMOUNT
1	106	3917.23
2	106	32138.68

Query 19

	CUST_ID	CUST_FNAME	CUST_LNAME	OQUANTITY
1	101	Andrew	Miller	3
2	105	Sam	Cooper	10
3	109	Bratt	Swift	100
4	106	Carmen	Powell	7

Query 20

	ORDER_DATE	SUM(I.INVOICE_AMOUNT)
1	12/01/2021	4241.33

Database Adminstration and Monitoring Roles and Responsibilities

• Database Administrator: The database administrator, and supporting database staff, shall oversee maintenance of the database and the development of new SQL scripts to support changing requirements.

• System Administrator: The system administrator and supporting staff shall maintain the state of the server running the DBMS, including the DBMS software itself, the server operating system, and any supporting tools.

• Security Administrator: The security administrator and other security staff shall maintain the integrity of the security measures and systems surrounding the database and will work directly with the other administration teams to oversee the upgrade of server software and the modification of the database and SQL scripts in responses to security issues and changes in security policy.

Data Formats.

The database, as currently designed, requires data transfer of three types: string, integer, time, and date data in the form of raw binary data; image transfer in the form of Portable Network Graphics (PNG) files; and level data transfer in the form of a proprietary level (.lvl) format used exclusively by the end user client. The raw binary data shall be stored in the database directly and transferred by the DBMS; while the storage and transfer of image and level data will be managed by a separate file storage system, which the database will link to via URIs pointing to specific files.

Backup and Recovery

Due to the expected frequency of changes made to the database in the form of new and updated level data, user registrations, and artifacts from end user interaction, delta backups of the database shall be performed twice daily, and a versioning system will temporarily store a record of changes as they are made. Full backups of the database will be performed during a weekly maintenance period at 3 AM EST every Tuesday.

END