PES UNIVERSITY

Department of Computer Science and Engineering

Course Code: UE23CS351A

Mini Project Report on

Fuel Management System

Submitted by:

Ritesh Reddy R S (PES1UG23CS482)

Section H

Under the guidance of:

Shruthi B P

Academic Year: 2025

ABSTRACT

A web-based database management system designed to automate fuel station operations, integrating real-time inventory, transaction management, and employee oversight for operational excellence and data integrity.

System Overview & Architecture

The Fuel Management System utilizes a robust client-server architecture with a relational database. It features a user-friendly web front-end and a back-end for data processing, business logic, and secure database communication, ensuring scalability and efficient real-time operations.

Purpose

Automate and streamline all core fuel station operations.

Scope

Real-time inventory, transaction processing, employee management, and comprehensive reporting.

Automation

Reduces errors, improves efficiency, and provides instant data access.

Core Entities

1

Fuel

Types, pricing, and current stock levels managed.

2

Transactions

Records of sales, purchases, and deliveries.

3

Employees

Roles, access controls, and performance tracking.

4

Customers

Loyalty programs and purchase history management.

5

Suppliers

Procurement and delivery schedule management.

Technology Stack

- **Backend:** Flask (Python)
- **Database:** Microsoft SQL Server
- **Connectivity:** ODBC Driver 17
- **Frontend:** HTML, CSS
- **Version Control:** GitHub
- **Query Language:** T-SQL

User Requirement Specification

Purpose

Automates essential fuel station operations (inventory, transactions, employee management), ensuring real-time updates, transparency, and data integrity via a relational database.

Scope

Supports fuel type, automated restocking, employee/pump management, and analytical reporting. Admin/employees interact via a Flask web interface with SQL Server.

Detailed Description

A Flask web application integrating with MS SQL Server (ODBC). Utilizes CRUD, stored procedures, triggers, and advanced queries for workflow automation. Features dashboards for inventory, transactions, and financial metrics.

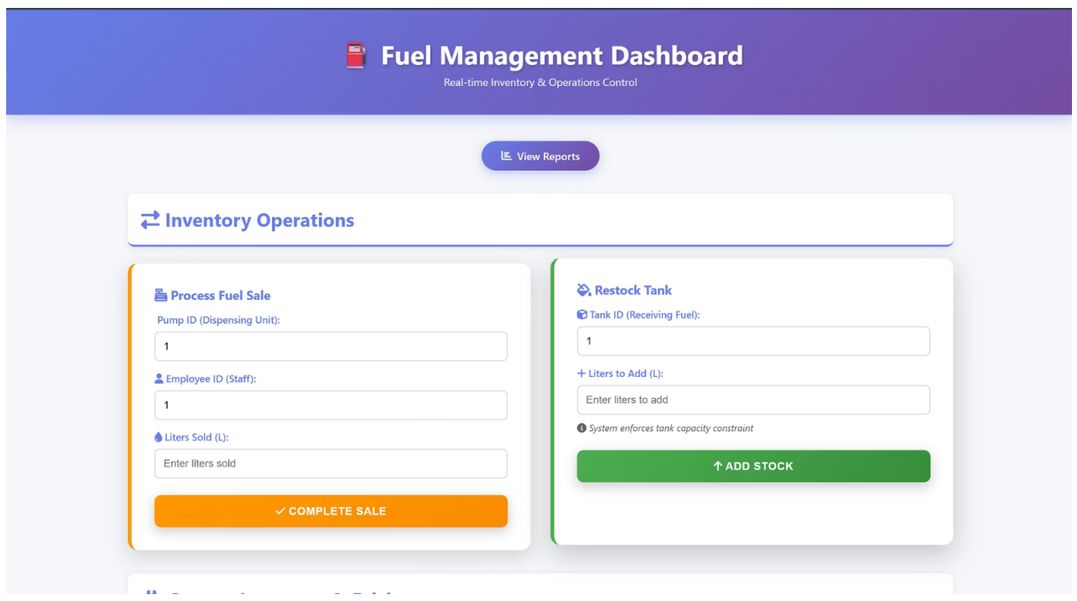
Functional Requirements

Functionality	Description
1. Manage Fuel Types	Add, edit, remove fuel types and prices.
2. Employee Management	Create, update, delete employee records with roles.
3. Transaction Recording	Log sales: employee, fuel, quantity, cost.
4. Restock Operations	Automated fuel tank restocking with overflow checks.
5. Dashboard Visualization	Display real-time tank levels and fuel prices.
6. Complex Queries	Generate join, nested, and aggregate analytical reports.
7. Error Handling	Validate inputs and rollback transactions on errors.

User Interface - Web Application

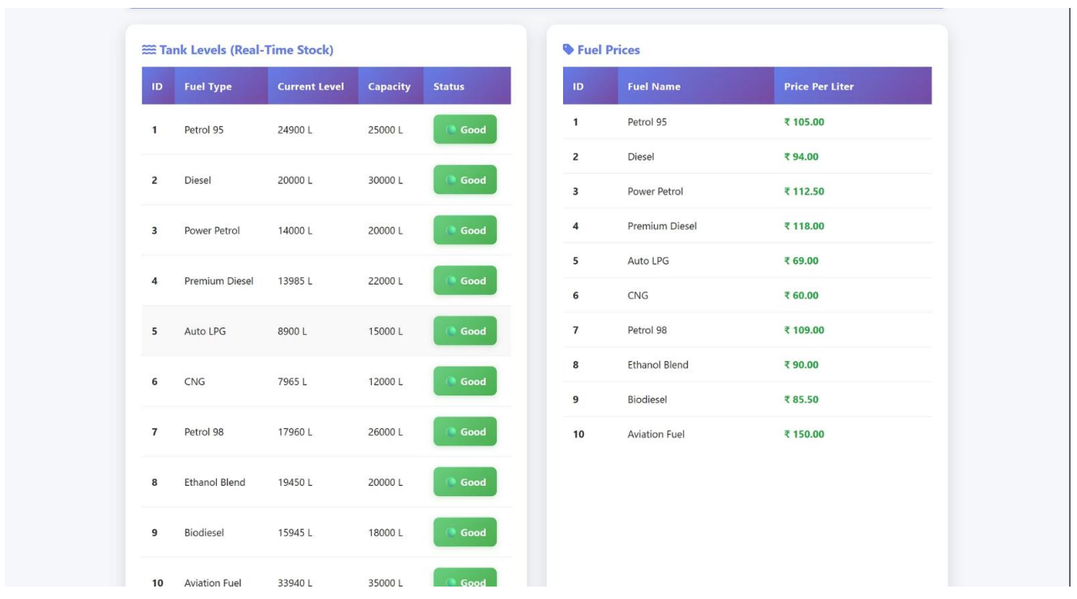
Dashboard Overview

This screenshot displays the main dashboard of the Fuel Management System, offering a quick glance at key operational metrics. It typically includes summaries of fuel levels, recent transactions, and system alerts for efficient monitoring.

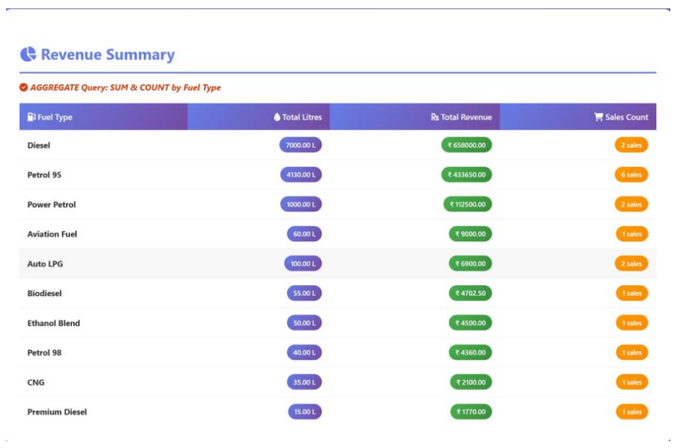


Fuel Price Management

This interface allows administrators to manage and update fuel types and their corresponding prices. It provides fields for entering new prices, adjusting existing ones, and viewing historical pricing data to ensure accuracy and timely updates.

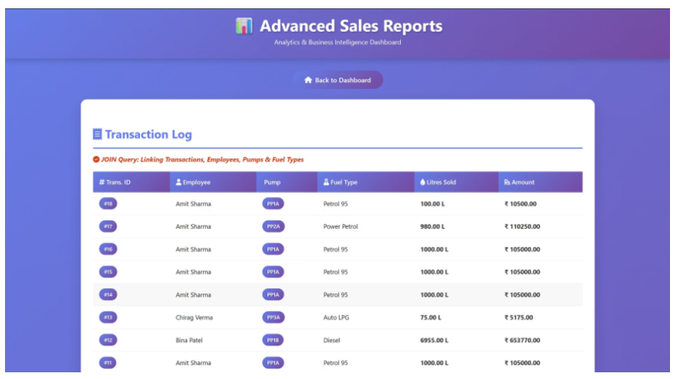


AGGREGATE QUERY



This screen provides a detailed log of all fueling transactions, including fuel type, quantity, date, time, and associated vehicle or driver. It allows for auditing and historical analysis of fuel consumption.

JOIN QUERY



Manage your entire fleet with ease. This interface allows you to add, edit, or remove vehicles, track their status, and assign them to specific drivers or departments, ensuring efficient resource allocation.

NESTED QUERY

This section is dedicated to managing driver profiles, including their credentials, assigned vehicles, and access permissions. It helps maintain accountability and streamline the fueling process for personnel.

CRUD Operations

This section demonstrates the fundamental Create and Read operations (CRUD) for managing employee data within the system.

Create Operation

```
INSERT INTO EMPLOYEES (Name, Role)
VALUES ('Rahul Mehta', 'Pump
Operator');
```

→ Adds a new employee record successfully.

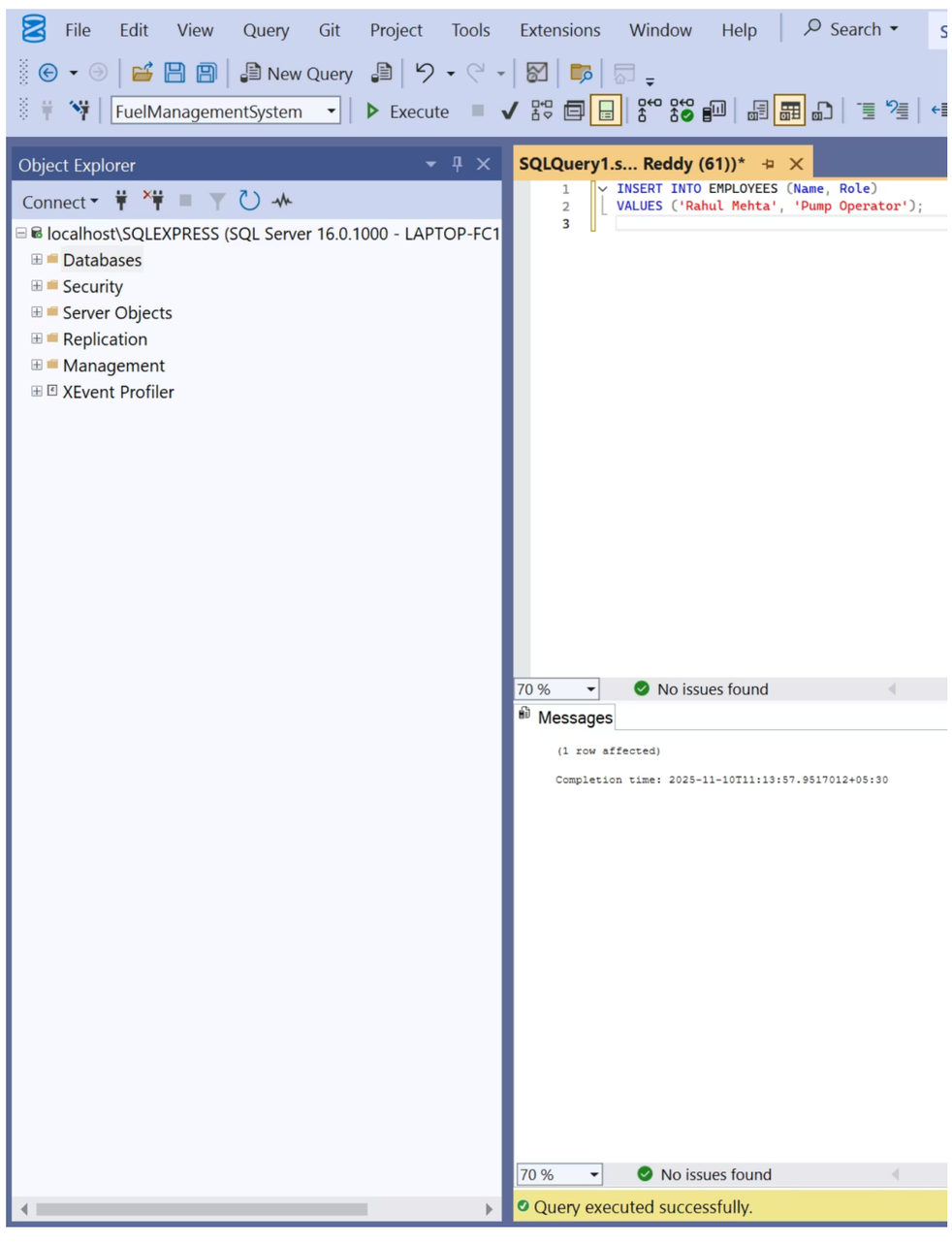
Query executed successfully

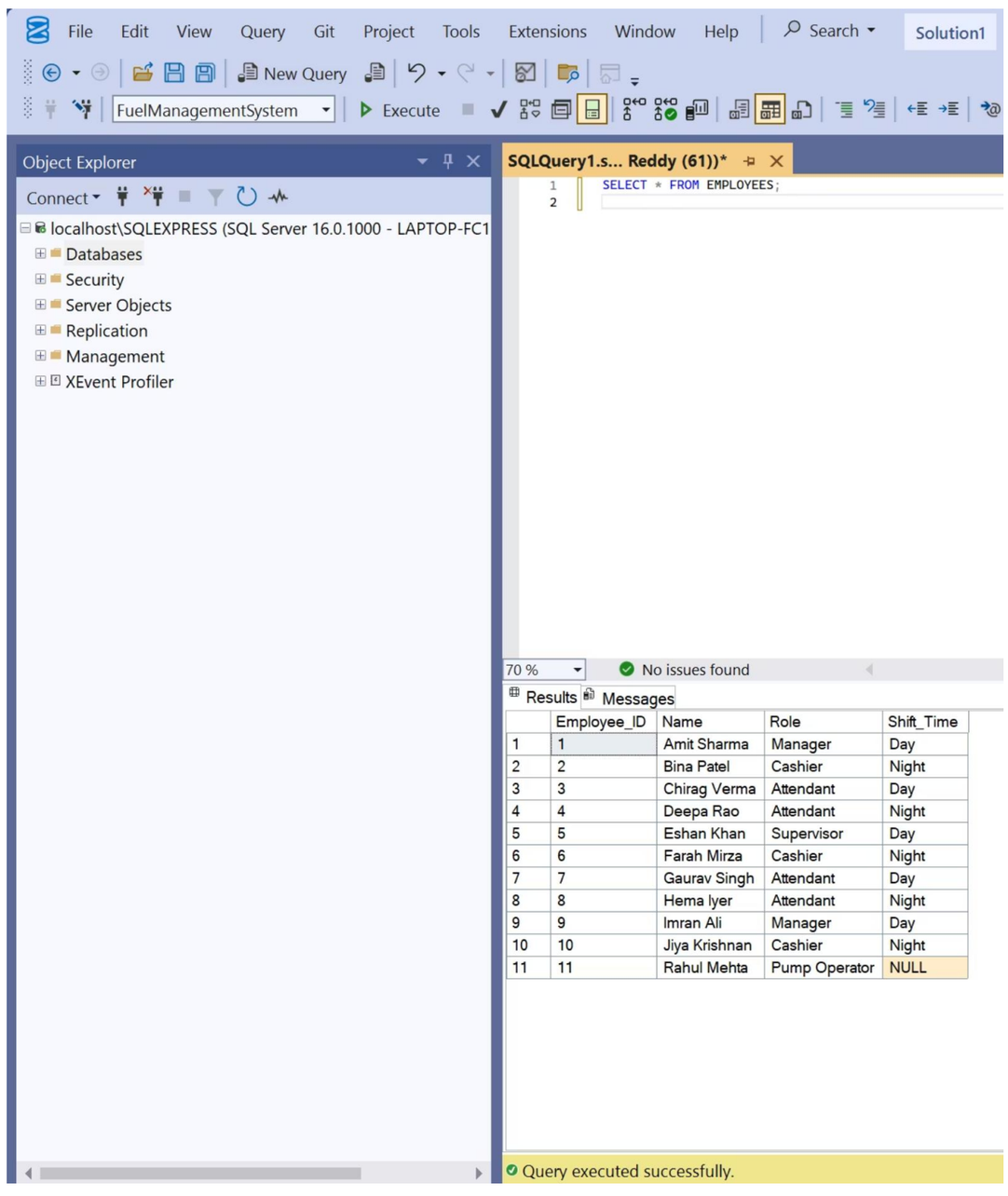
Read Operation

```
SELECT * FROM EMPLOYEES;
```

→ Displays all employee records including the new entry.

Query executed successfully





	Employee_ID	Name	Role	Shift_Time
1	1	Amit Sharma	Manager	Day
2	2	Bina Patel	Cashier	Night
3	3	Chirag Verma	Attendant	Day
4	4	Deepa Rao	Attendant	Night
5	5	Eshan Khan	Supervisor	Day
6	6	Farah Mirza	Cashier	Night
7	7	Gaurav Singh	Attendant	Day
8	8	Hema Iyer	Attendant	Night
9	9	Imran Ali	Manager	Day
10	10	Jiya Krishnan	Cashier	Night
11	11	Rahul Mehta	Pump Operator	NULL

CRUD Operations

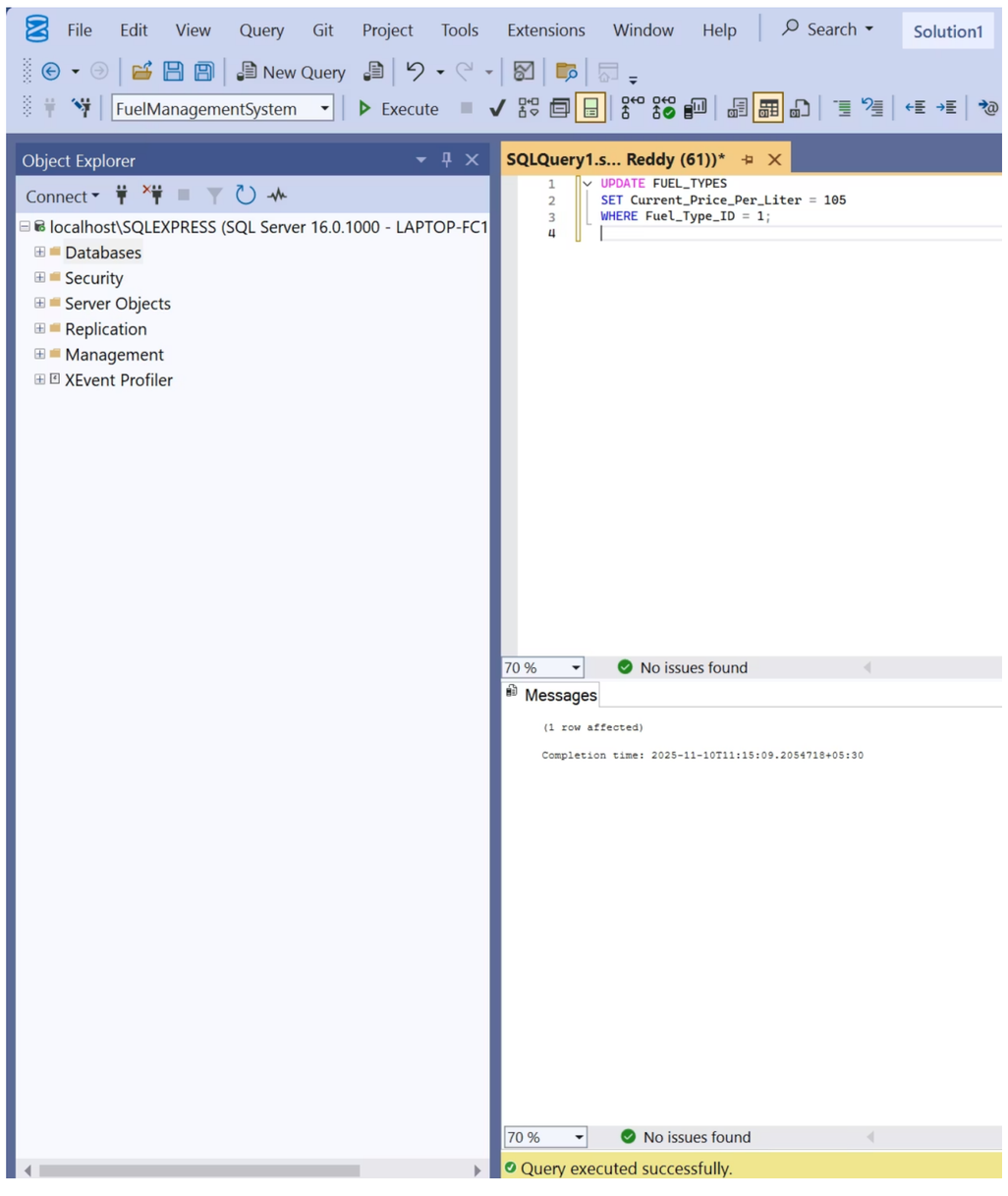
This section continues our exploration of CRUD operations, focusing on how data within the system is updated and deleted to maintain accuracy and relevance.

Update Operation

```
UPDATE FUEL_TYPES SET
Current_Price_Per_Liter = 105 WHERE
Fuel_Type_ID = 1;
```

→ Updates the fuel price for Petrol.

Query executed successfully

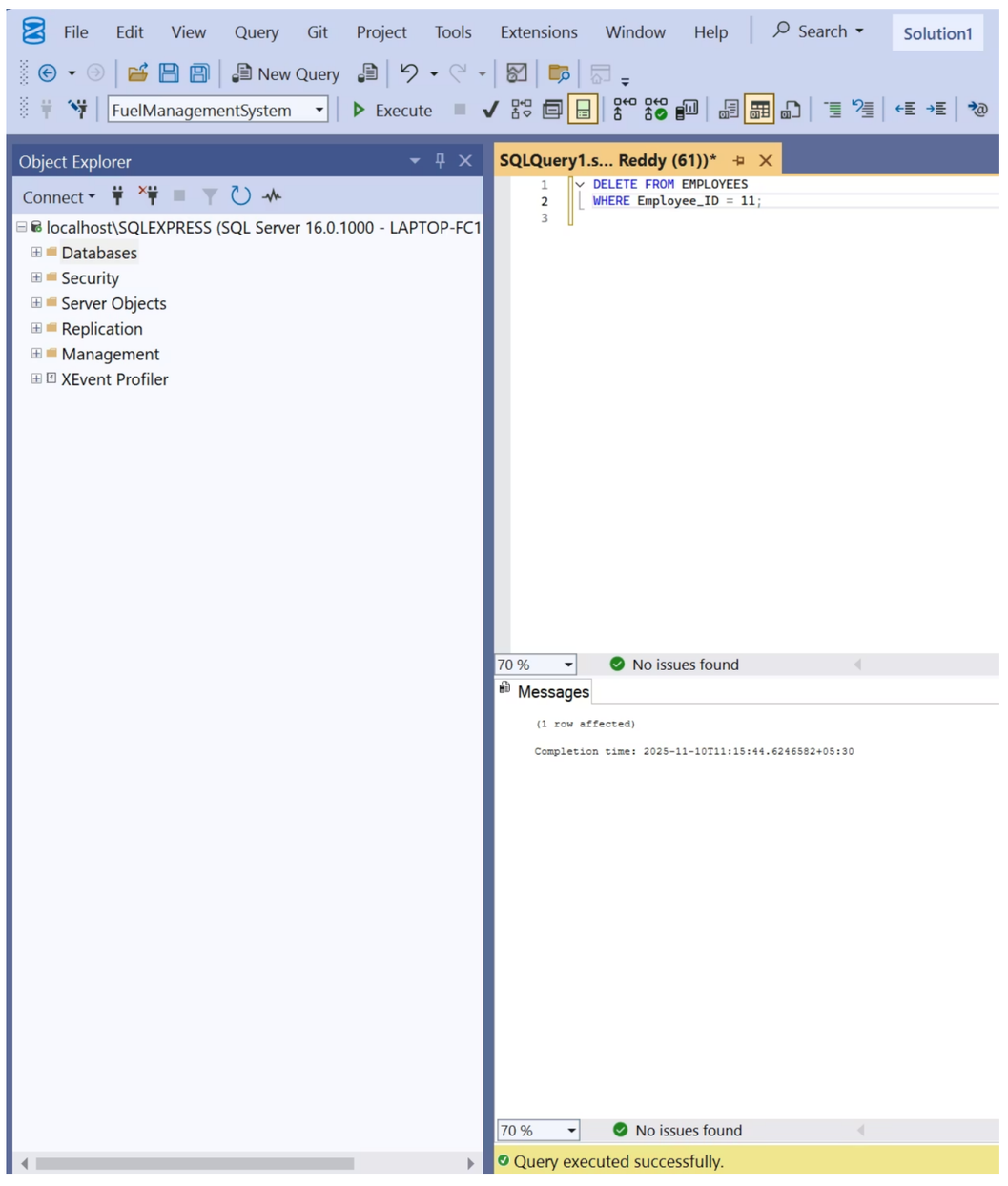


Delete Operation

```
DELETE FROM EMPLOYEES WHERE
Employee_ID = 11;
```

→ Removes the specified employee record.

Query executed successfully



These CRUD operations demonstrate the system's ability to manage data effectively, ensuring data integrity and providing complete database functionality for the Fuel Management System.

Database Design & Implementation

Relational Schema Architecture

The system utilizes a normalized relational schema with integrity constraints. Primary and foreign keys ensure referential integrity. Stored procedures `ProcessSale()` and `RestockTank()` manage core business logic, while triggers automate tank level updates, ensuring data consistency.

01

Data Definition Language Complete DDL with PK, FK, and unique constraints for data integrity.

02

Stored Procedures Encapsulated business logic for sales and restocking with transactional guarantees.

03

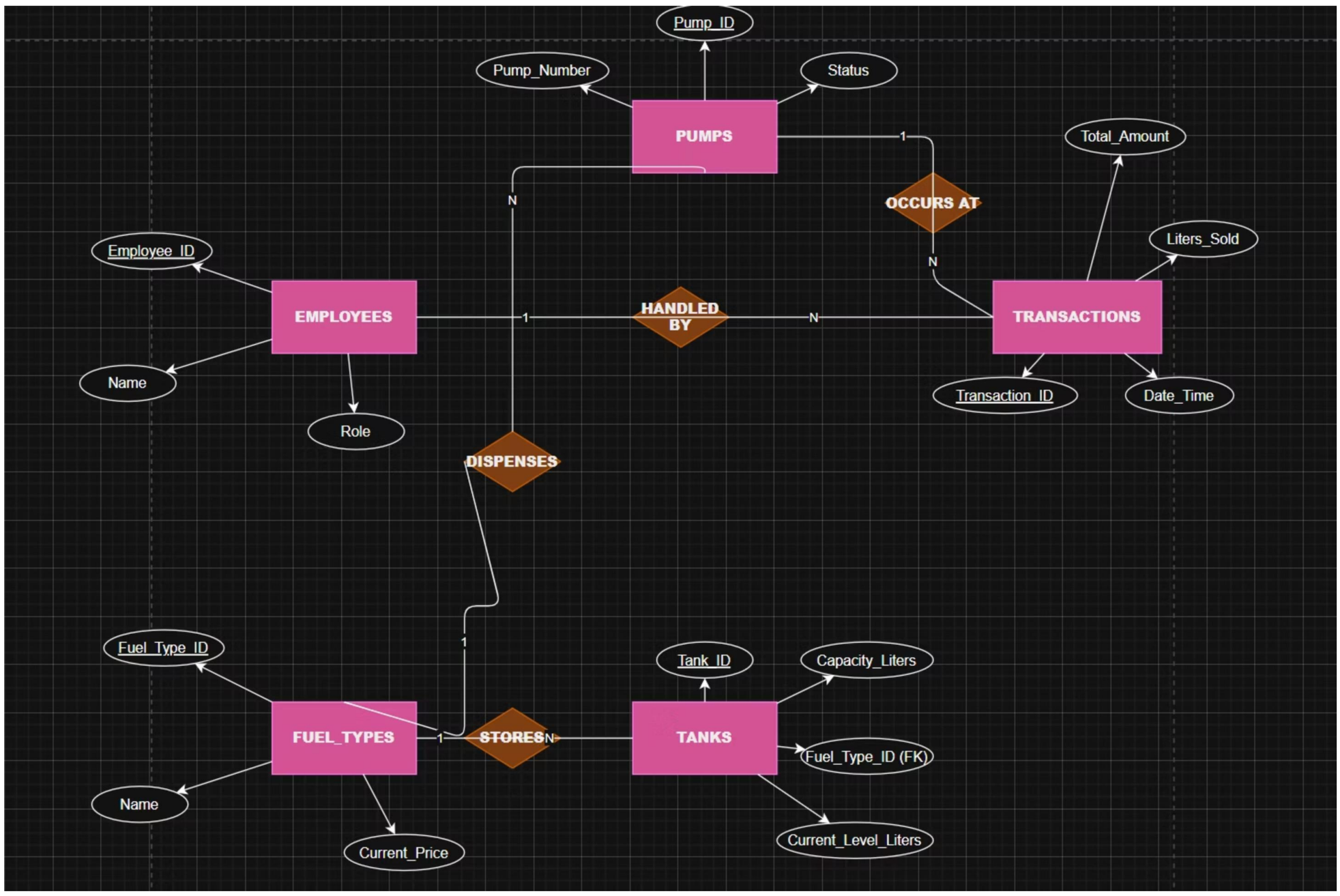
Trigger-Based Automation Real-time tank level updates triggered automatically on transactions.

04

Advanced Querying Complex SQL queries (JOINs, aggregates, subqueries) for reporting.

Entity-Relationship Diagram

Visual representation of core entity relationships.



Detailed Code Snippets

Stored Procedure: ProcessSale

```
CREATE PROCEDURE ProcessSale     @PumpID INT, @EmployeeID INT, @LitersSold
FLOATASBEGIN    DECLARE @PricePerLiter FLOAT;    INSERT INTO TRANSACTIONS (Pump_ID,
Employee_ID, Liters_Sold, Total_Amount, [datetime])    VALUES (@PumpID,
@EmployeeID, @LitersSold, @LitersSold * @PricePerLiter, GETDATE());END;
```

Handles sales, calculates total.

Stored Procedure: RestockTank

```
CREATE PROCEDURE RestockTank     @TankID INT, @LitersAdded FLOATASBEGIN    DECLARE
@Capacity FLOAT, @Current FLOAT;    IF (@Current + @LitersAdded > @Capacity)
RAISERROR('Capacity exceeded.', 16, 1);    ELSE        UPDATE TANKS SET
Current_Level_Liters = @Current + @LitersAdded;END;
```

Updates tank levels, checks capacity.

Trigger: UpdateTankLevel

```
CREATE TRIGGER trg_UpdateTankLevel ON TRANSACTIONS AFTER INSERTASBEGIN    UPDATE T
SET T.Current_Level_Liters = T.Current_Level_Liters - i.Liters_Sold    FROM TANKS T
INNER JOIN PUMPS P ON T.Fuel_Type_ID = P.Fuel_Type_ID;END;
```

Auto-updates tank levels post-sale.

Function: GetTotalRevenue

```
CREATE FUNCTION GetTotalRevenue()RETURNS FLOATASBEGIN    DECLARE @Total FLOAT;
SELECT @Total = SUM(Total_Amount) FROM TRANSACTIONS;    RETURN @Total;END;
```

Retrieves total revenue.

Flask Integration Example

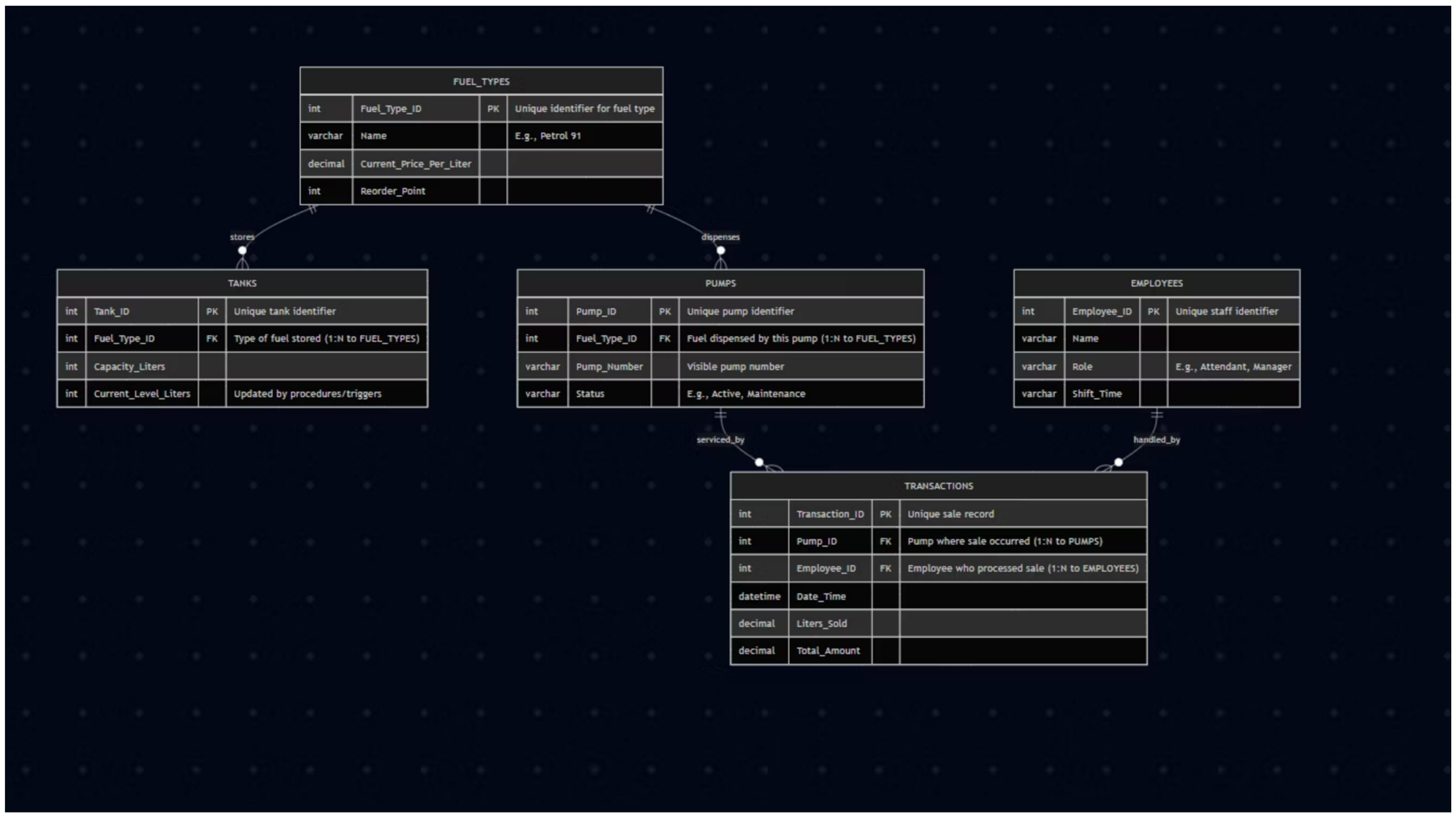
```
@app.route('/process_sale', methods=['POST'])def process_sale():    """Calls the
ProcessSale stored procedure."""    conn = get_db_connection()    cursor =
conn.cursor()    cursor.execute("{CALL dbo.ProcessSale (?, ?, ?)}",
pump_id, employee_id, liters_sold)    conn.commit()    flash("Sale processed
successfully! Inventory updated by Trigger.", 'success')
```

Flask integration with SQL stored procedures.

Key Implementation Features

- **Stored Procedures:** Encapsulate sales & restocking logic
- **Triggers:** Automatic inventory updates for data consistency
- **Functions:** Enable calculations & data retrieval
- **Flask Integration:** Connects web interface to database operations
- **Error Handling:** Validation & rollback for data integrity
- **Real-time Updates:** Automatic sync of transactions & inventory

Database Relationship Schema



Key Capabilities & Features

The system delivers comprehensive functionality designed for efficient fuel station management. Through real-time dashboards, automated transaction processing, and intelligent reporting, the platform transforms operational workflows and provides actionable business intelligence.

01

Real-Time Dashboard

Live visualization of tank levels and fuel pricing.

02

Transactional Processing

Validated SQL procedures for sales and restocking, with full audit trails.

03

Automated Inventory Updates

Database triggers for automatic stock level updates.

04

Advanced Reporting

Sophisticated queries for business insights.

05

Role-Based Management

Role-based employee access control.

Results & Conclusion

Operational Excellence

Automated fuel tracking and stock updates eliminate manual processes and reduce operational overhead

Data Accuracy

Systematic error reduction through trigger-based consistency and validated transaction processing

Business Intelligence

Analytical insights enabling informed management decisions through comprehensive reporting capabilities

System Impact

The Fuel Management System successfully demonstrates core DBMS concepts through practical implementation. By integrating relational design principles, stored procedures, and automated triggers, the system streamlines daily fuel station operations while maintaining data integrity. This project exemplifies how sophisticated database architecture transforms manual workflows into efficient, scalable business processes.

GitHub Repository: https://github.com/riteshreddyyy/Fuel-Management-System