Final Project Submission – Shopfloor Inventory Management

MIS 659

Relational Database Management

Mr. David Sanford

Shopfloor Inventory Management System

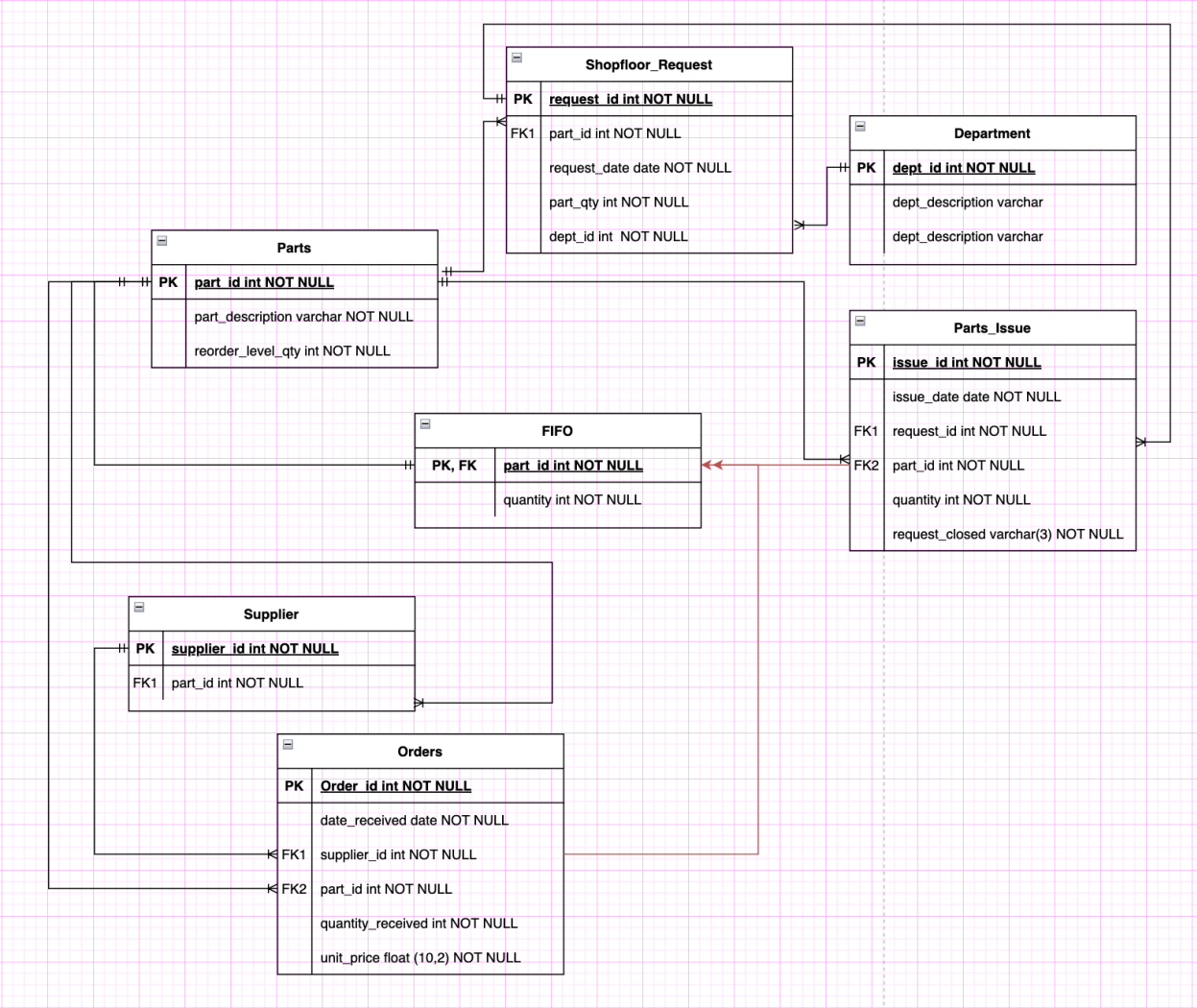
This document details the process of designing a database structure to manage inventory of parts for a shopfloor that is engaged in assembling home appliances such as refrigerators and air conditioners. This hypothetical shopfloor consists of few Assembly lines (please note that initially I have inserted data for four assembly lines and to test user interface created with Python code using Qt5 Python Widgets I have input more Assembly lines). A sample of ten types of parts are taken for this assignment. The iterative process of the system involves obtaining request for material parts from assembly lines, issuing parts to respective assembling departments, ordering the parts with pre-approved vendors as and when reorder levels are reached, automatically updating inventory table with available parts of each variety upon issue of parts as well as orders received from vendors.

Database Structure

The database structure consists of seven interrelated tables to maintain data integrity and consistency viz. Department (Assembly lines), Parts, Shopfloor_Request, Supplier, Orders, Parts_Issue, and FIFO (First-in-First-Out inventory). Each of the rows of these database tables store unique records. The database tables viz. Parts and Department are independent tables with no foreign keys. Null values are restricted for all the tables. Relationships between these database tables along with an ER diagram are described in next section.

Database Design

Figure 1: ER Diagram



In this section the relationships between the tables and key constraints are discussed.

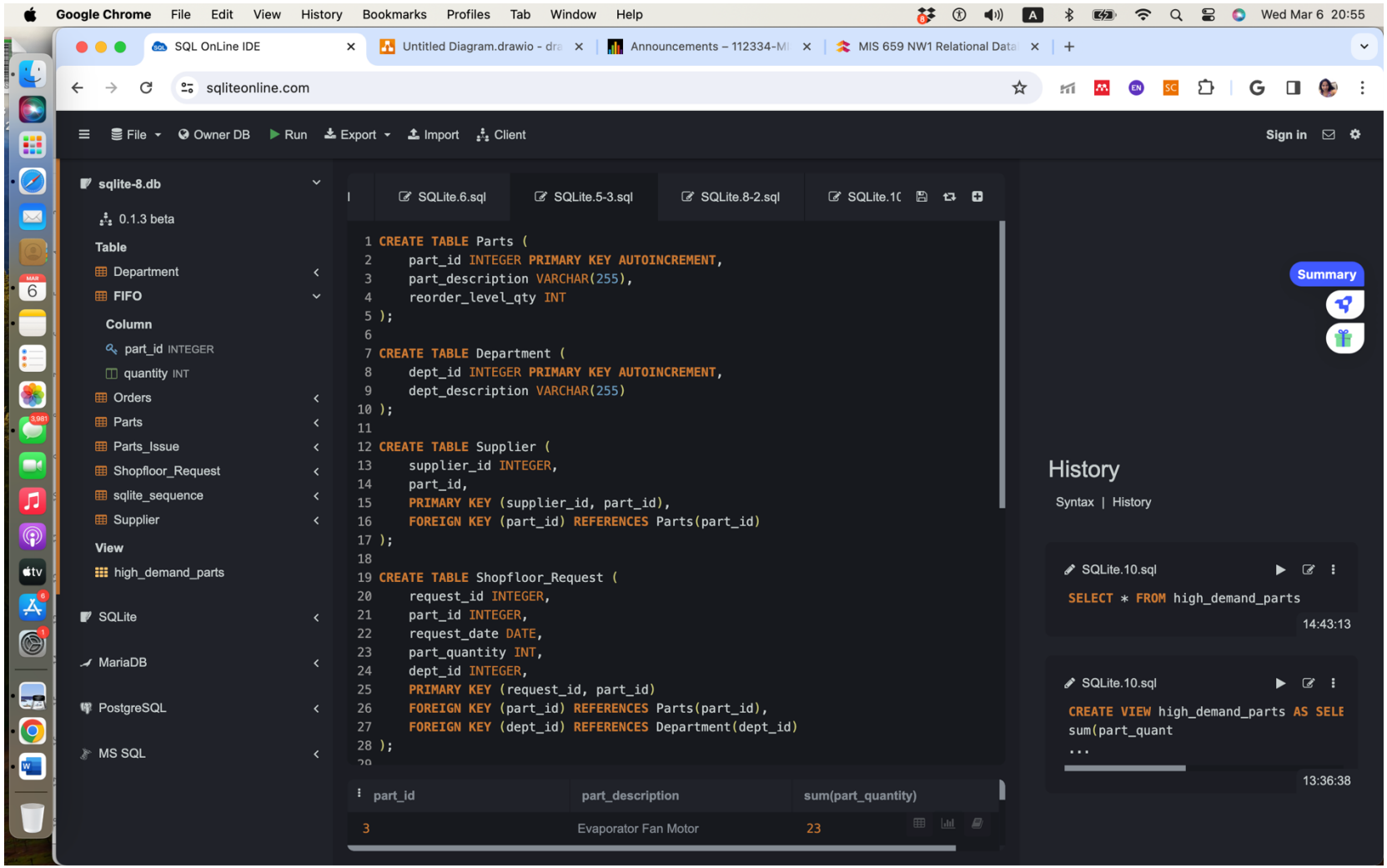
- There is a one-to-many relationship between Department and Shopfloor_Request, a department table can have unique departments but could create multiple requests for material parts.
- FIFO table has a one-to-one relationship with Parts table and as the relationship line indicates a minimum of one part_id is necessitated for both tables.

- part_id is primary as well as foreign key for FIFO table. As denoted with colored lines in the above ER Diagram, FIFO table is automatically updated upon issue of parts and receipt of orders (using triggers).
- Another trigger is created to update the field “request_closed” as and when entire quantity of parts of a request_id are issued.
- Parts table is one of the crucial link between tables with five other tables depending on it.
- Please note that as indicated in the cardinality and connectivity lines, a minimum of one entry for primary and foreign key fields is maintained.
- Database Views are created to query the most sought parts by assembly lines.
- Other relationships in the above ER diagram are self-explanatory.

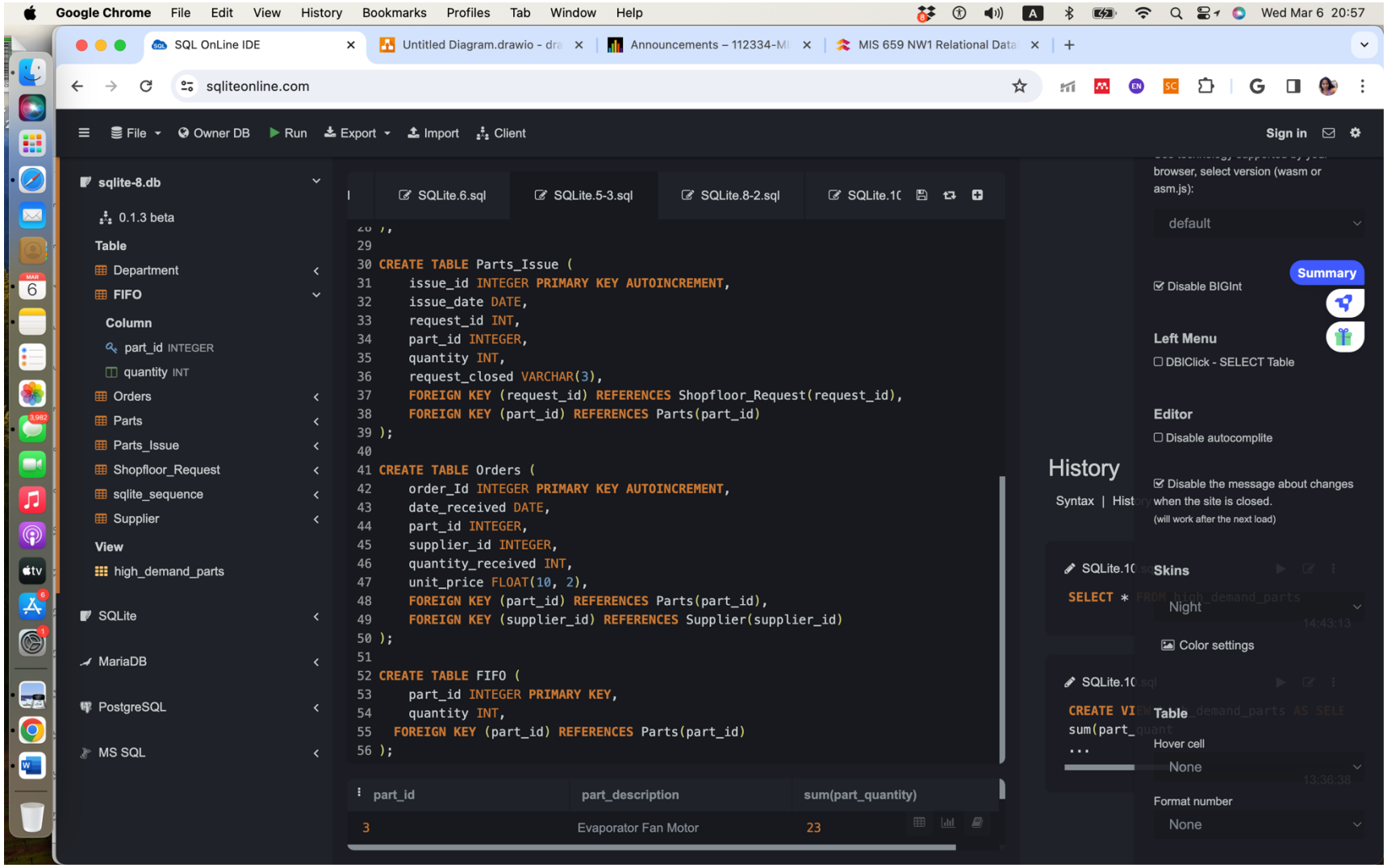
Final Project Submission – Shopfloor Inventory Management

Table Creation and Data types

Here are the screenshots of the creation of database tables (DB file and sql files are attached/submitted along with this document).



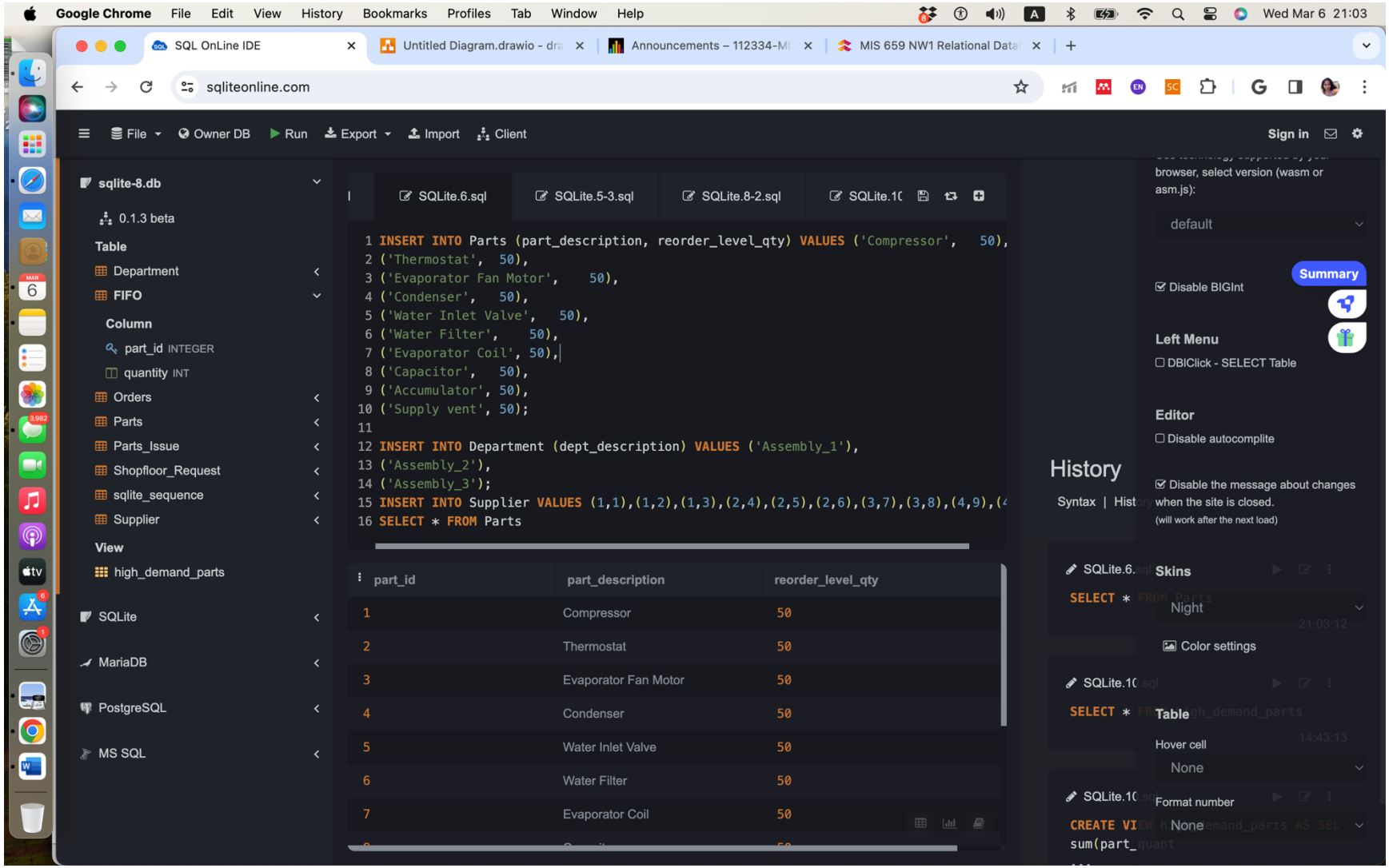
Final Project Submission – Shopfloor Inventory Management



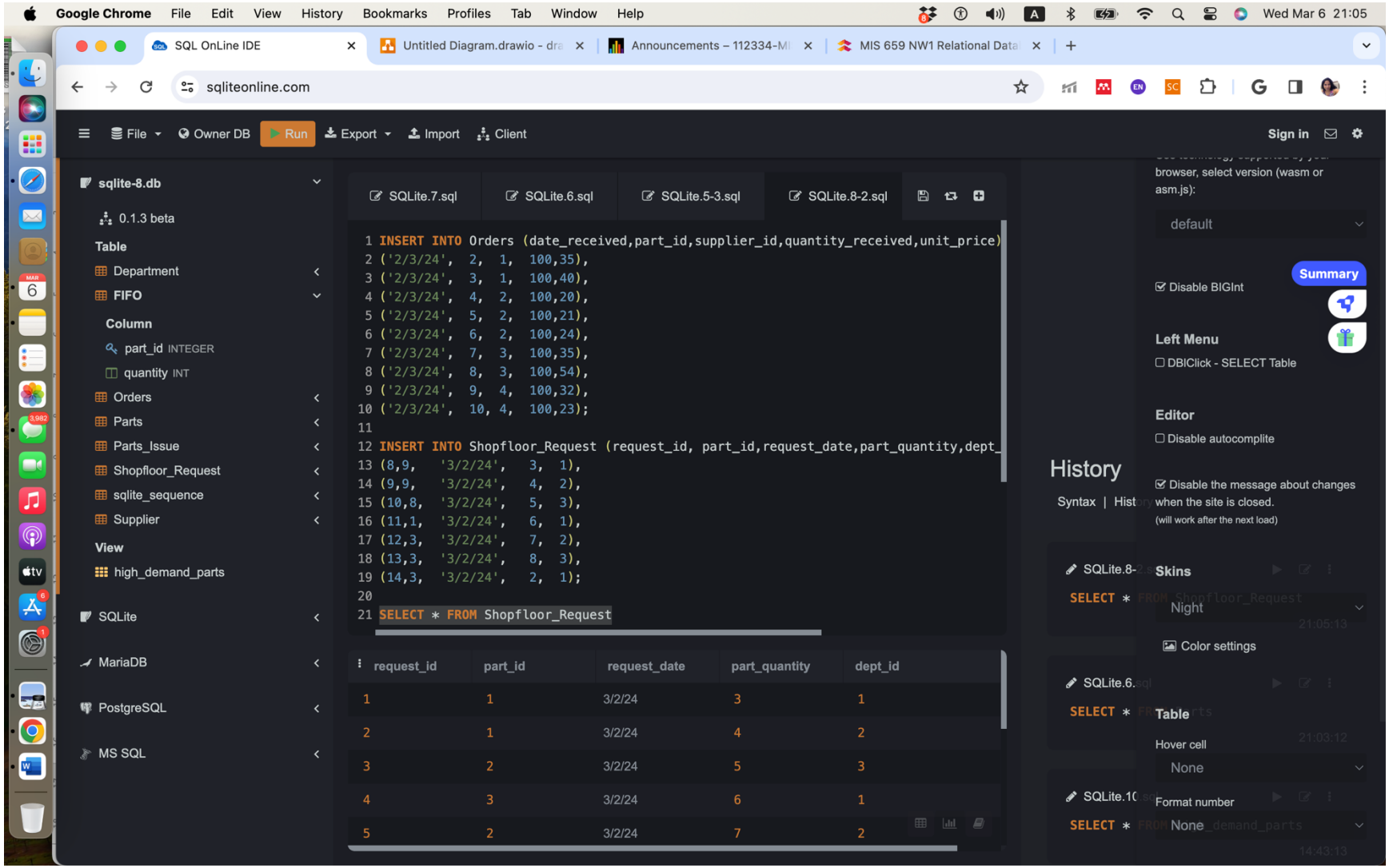
Final Project Submission – Shopfloor Inventory Management

Data Population

Screenshots for data input are attached below.



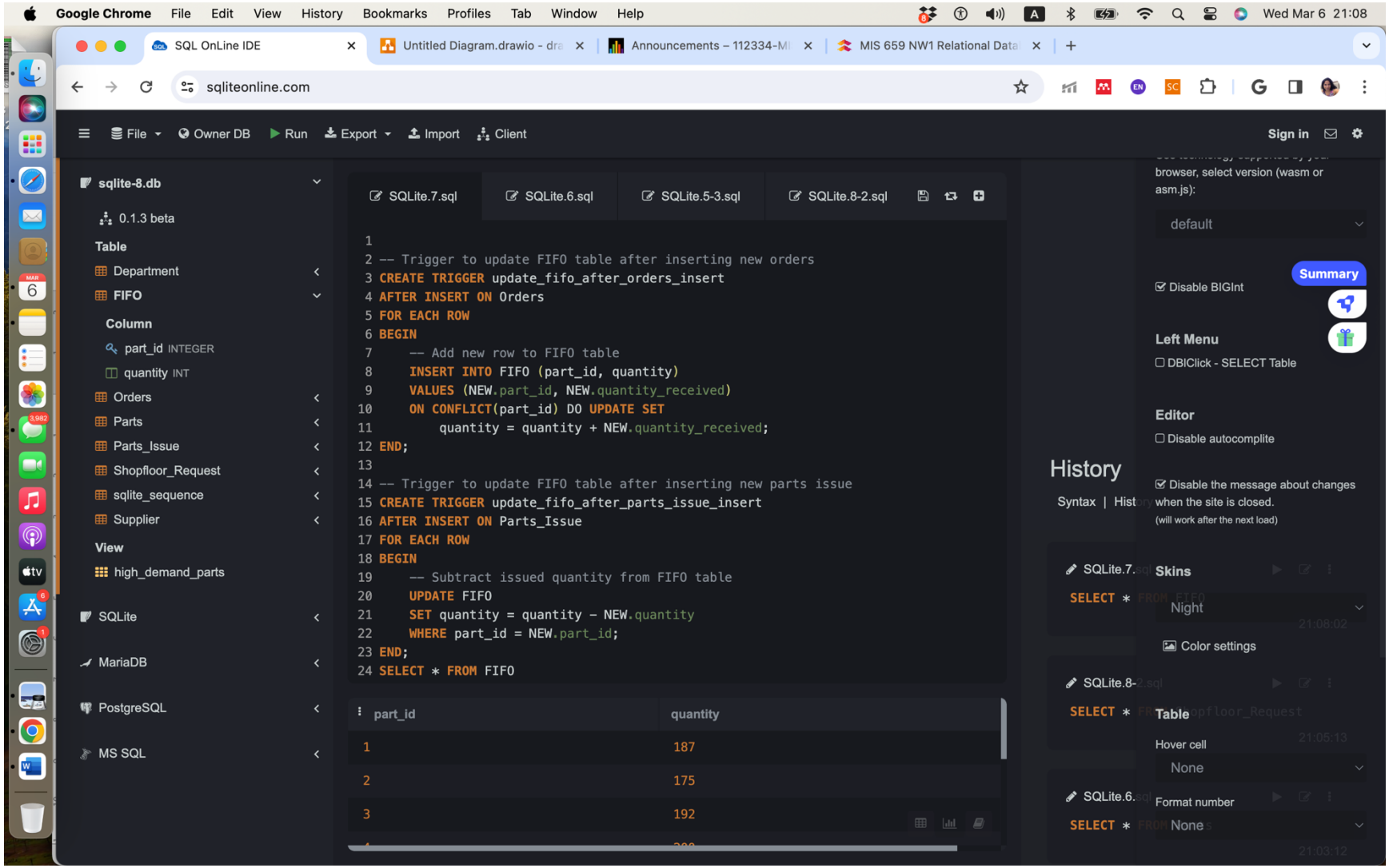
Final Project Submission – Shopfloor Inventory Management



Final Project Submission – Shopfloor Inventory Management

Triggers and Complex Queries

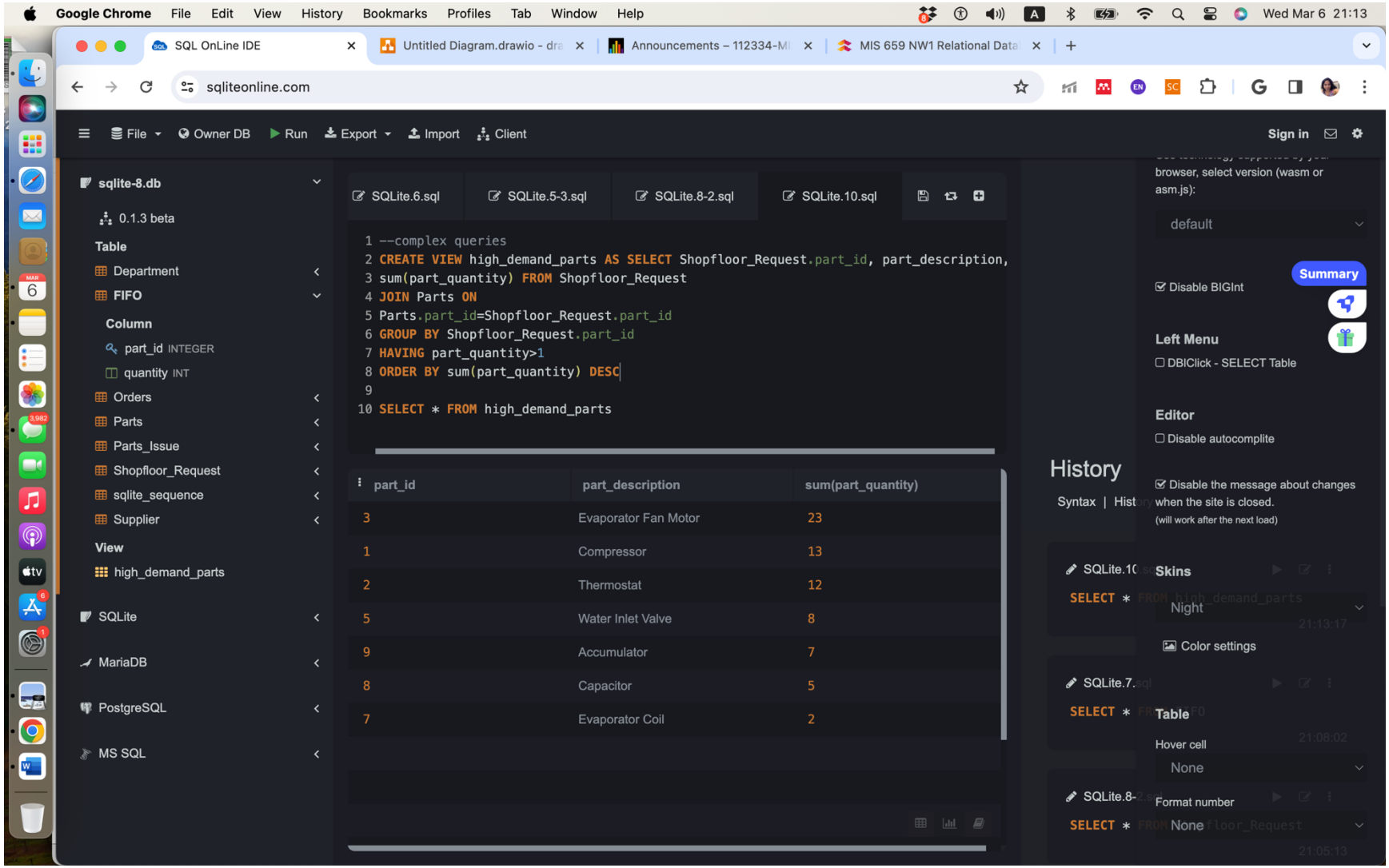
As described earlier, these triggers are run immediately after insertion into Orders table and Parts_Issue table and automatically update quantities of parts in FIFO table.



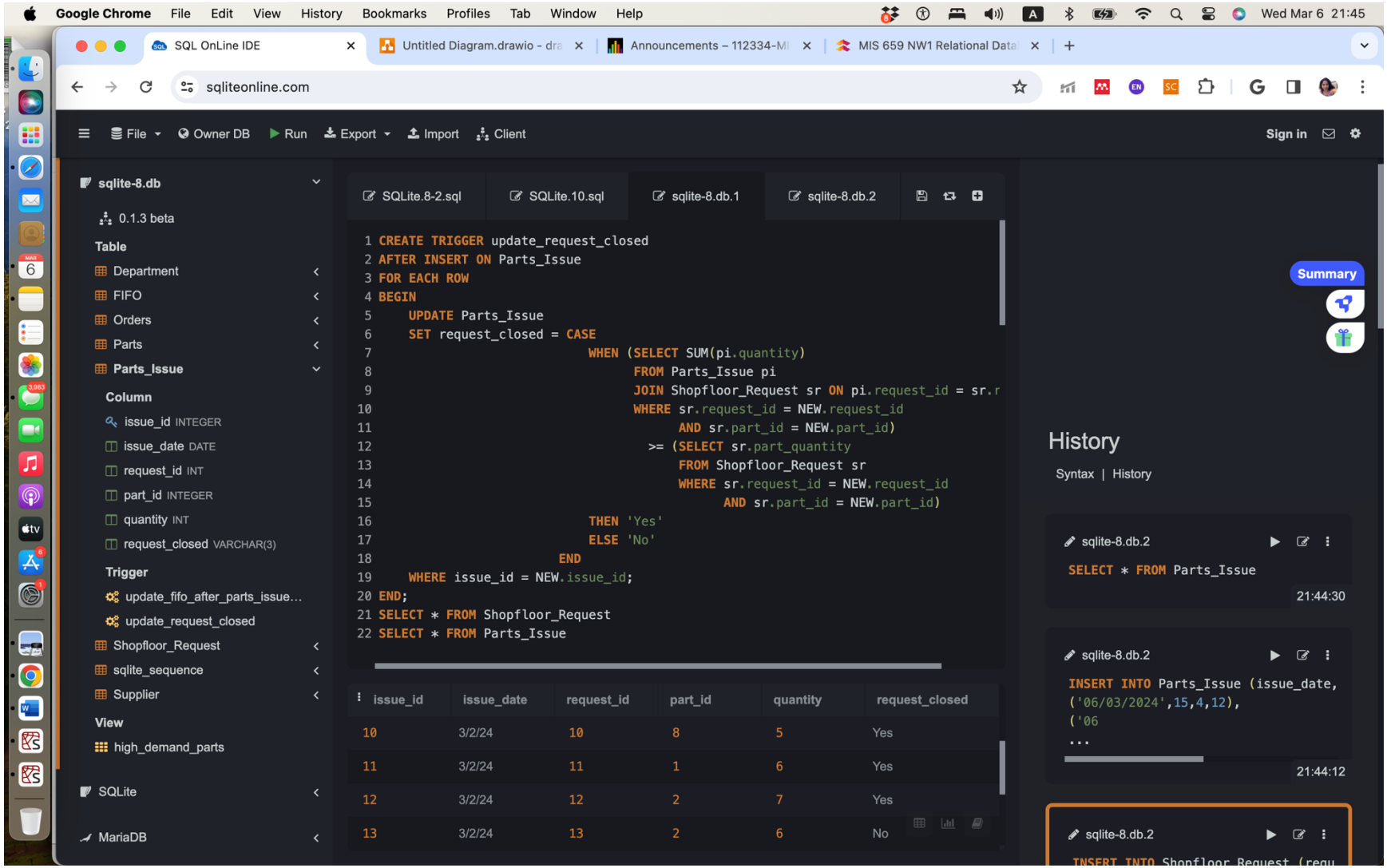
Final Project Submission – Shopfloor Inventory Management

Complex Query – Creation of View

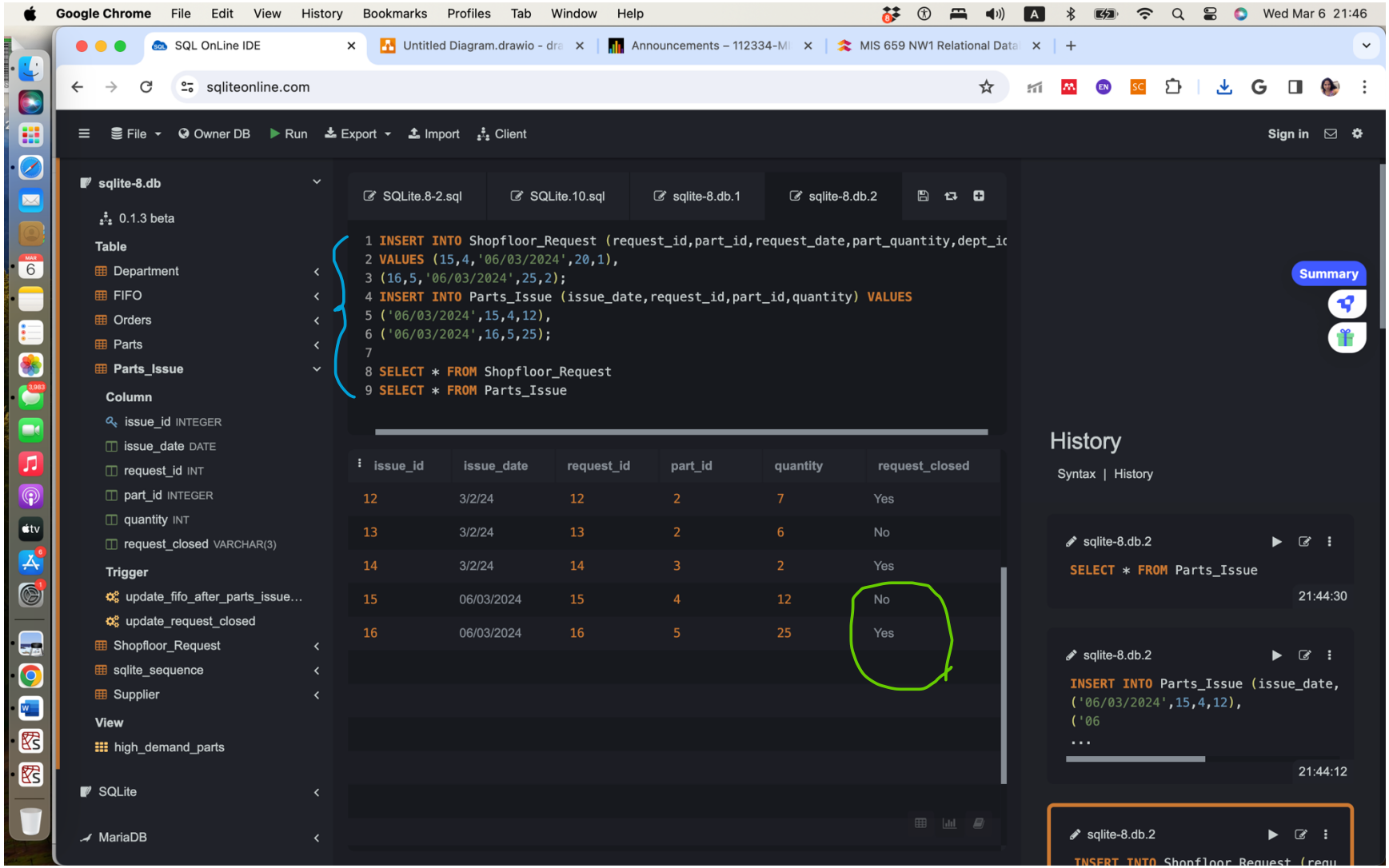
Running this query gives information about high demand parts from Assembly Lines



Trigger to update the field ‘request_closed’

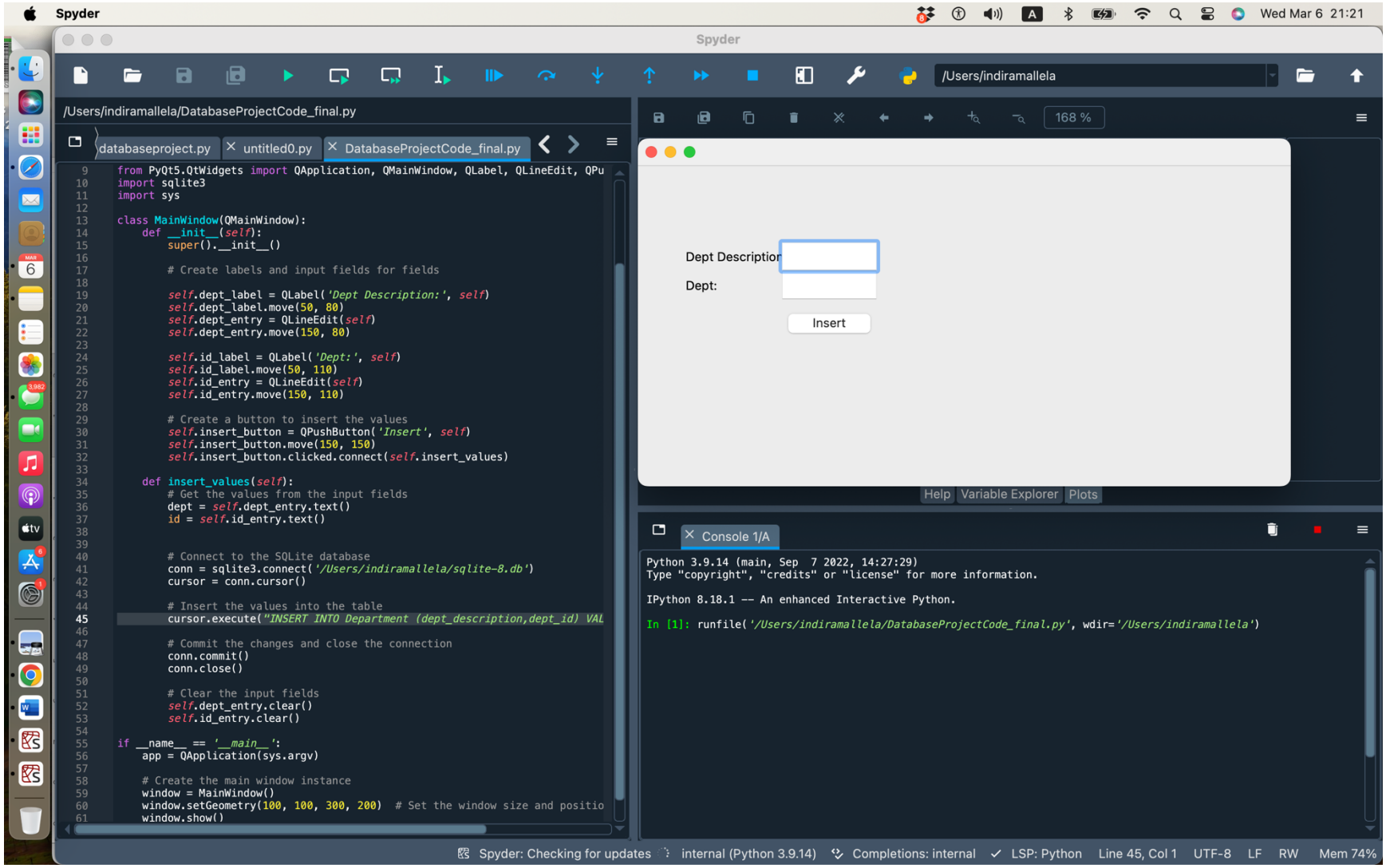


Final Project Submission – Shopfloor Inventory Management



Final Project Submission – Shopfloor Inventory Management

User Interface using Python QWidgets (Python Code document is also submitted)




```
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel, QLineEdit, QPu
import sqlite3
import sys

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # Create labels and input fields for fields

        self.dept_label = QLabel('Dept Description:', self)
        self.dept_label.move(50, 80)
        self.dept_entry = QLineEdit(self)
        self.dept_entry.move(150, 80)

        self.id_label = QLabel('Dept:', self)
        self.id_label.move(50, 110)
        self.id_entry = QLineEdit(self)
        self.id_entry.move(150, 110)

        # Create a button to insert the values
        self.insert_button = QPushButton('Insert', self)
        self.insert_button.move(150, 150)
        self.insert_button.clicked.connect(self.insert_values)

    def insert_values(self):
        # Get the values from the input fields
        dept = self.dept_entry.text()
        id = self.id_entry.text()


        # Connect to the SQLite database
        conn = sqlite3.connect('/Users/indiramallela/sqlite-8.db')
        cursor = conn.cursor()

        # Insert the values into the table
        cursor.execute("INSERT INTO Department (dept_description,dept_id) VAL

        # Commit the changes and close the connection
        conn.commit()
        conn.close()

        # Clear the input fields
        self.dept_entry.clear()
        self.id_entry.clear()

if __name__ == '__main__':
    app = QApplication(sys.argv)

    # Create the main window instance
    window = MainWindow()
    window.setGeometry(100, 100, 300, 200)  # Set the window size and positio
    window.show()
```

Conclusion

This project has helped me gain insights into how database systems are designed, and constraints are built to maintain robustness of data. My understanding on triggers, views, and complex queries has enhanced significantly. This exercise gave me hands on experience to work on real world inspired project and explore ways to automate the processes in database management.