**Solent University**

Advanced Database Systems

**Module Leader**:

Dr. Muhammad Akram

Project with Report

**Student ID**: [Your ID Number]

**Student Name**: [Your Full Name]

**Submission Date**: 4th February 2025

**Word Count**: ???

Table of Contents

**Introduction**

This report describes the construction of a database system for a restaurant. The main aim of the project was to construct a highly efficient and user-friendly database-based application, which can perform all day-to-day activities in the restaurant as smoothly as possible e.g. making table reservations, tracking sales, managing inventory, handling orders, etc.

Managing data in restaurants is still one of the biggest challenges. Most medium and small-scale restaurants use half-manual or not-so-advanced computer applications to manage their data, and thus errors and duplication are not uncommon. Something like this should not happen because it may cause loss of important information about customers i.e. missing out on dollars from opportunities.

The objectives of this report are to describe the client’s requirements, design an efficient relational database, implement a proof-of-concept system and evaluate its performance for the solution of the problems defined above. SQLite is used as a lightweight but powerful database management system that will be encapsulated by Tkinter, Python’s standard GUI (Graphical User Interface) package. XML will be used for storage and manipulation of arbitrary structured data.

The database is designed with third normal form (3NF) to make data consistency and prevent data for redundancy, relational features like Triggers, Views are also constructed to make system works effectively and efficiently as business wants, as well as usability. And finally the tested results which is performed by users of system was shown to include the context of the functional requirements that have been selected.

In conclusion, our database system for restaurant management shows how using a well-designed database could ease daily operations and save expenses by minimizing errors.

**Part 1: Client Requirements and Database Design** **outline**

This section outlines the essential client requirements for the database system, followed by the process of designing and creating the database using SQLite. The goal is to create a system that meets the needs of the restaurant, ensuring it can handle customer data, inventory, employee information, and order details effectively.

**1.1. Business Case for the Chosen Restaurant**

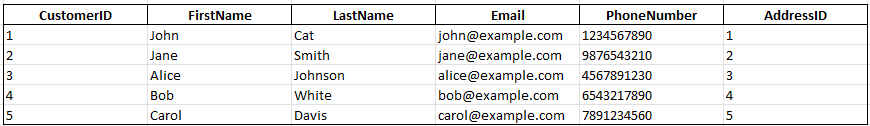
The restaurant is a small-sized bistro specializing in Italian cuisine, with both dine-in and delivery services. It employs approximately 10 staff members and serves around 50 customers daily.

**1.2. Seven Business Requirements**

1. **Customer Management** – The system should allow for storing and retrieving customer information, including personal details like address and phone number.
2. **Inventory Tracking** – The system should track inventory levels for food ingredients and supplies.
3. **Employee Records** – The database should maintain employee information, such as contact details, job title, and payroll information.
4. **Reservation Management** – The system must handle customer reservations, ensuring no double-booking occurs.
5. **Financial Records** – The database should track sales, revenue, and expenses, including generating financial reports.
6. **Menu Management** – The system should allow for managing and updating menu items, including prices and availability.
7. **Order Management** – The system should manage customer orders, order details and accurate billing.

**2. Flat File Creation with Collected Data (Excel)**

For initial data collection, a flat file was created using Microsoft Excel. Below is a screenshot of the flat file created for the 'Customers' table, which includes five records. Similar flat files were created for other tables in the database, including 'Suppliers', 'Reservations’ and 'Staff'. Each file contains sample data to be used in the SQLite database development.

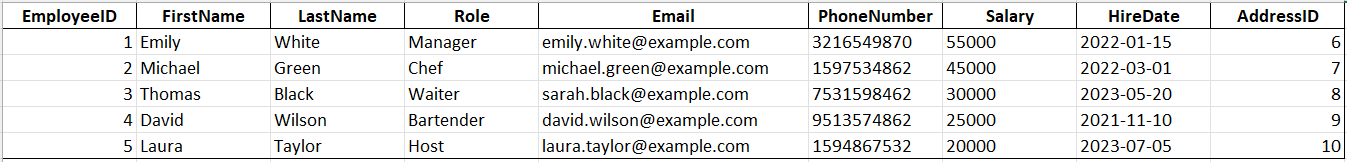


**Customers**

Obraz zawierający tekst, Czcionka, numer, linia

Opis wygenerowany automatycznie

**Suppliers**

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

**Reservations**

**Staff**

**3. Relational Schema**  **and Database Design**

In this section, we present the relational schema for the restaurant management database system, represented through an Entity-Relationship Diagram (ERD). By adhering to the principles of Third Normal Form (3NF), the design minimizes redundancy and ensures consistency within the database.

The ERD below illustrates the key entities, their attributes, and relationships:

Obraz zawierający tekst, zrzut ekranu, numer, design

Opis wygenerowany automatycznie

**Justification for 3NF Compliance**

To ensure that the database is normalized to Third Normal Form (3NF), the following steps and principles were followed:

**First Normal Form (1NF)**

**Criteria:**

* Each column must contain atomic values (no multiple values in one column).
* Each row must be unique, and there must be a primary key.

**Analysis:**

* All tables have clearly defined primary keys (e.g., CustomerID, OrderID, AddressID, etc.).
* All columns store atomic values (no arrays, lists, or multiple values in a single column).
* There is no redundancy or repetition of groups of data in single rows.

**Second Normal Form (2NF)**

**Criteria:**

* The database must meet 1NF.
* All non-primary-key attributes must be fully functionally dependent on the entire primary key (no partial dependency).

**Analysis:**

* In tables with composite primary keys (e.g., MenuIngredients), non-primary attributes depend on the entire primary key (MenuItemID, IngredientID).
* For tables with single-column primary keys (e.g., Orders, Customers), there are no attributes dependent on only part of the key, as all non-primary attributes relate fully to the primary key.

**Third Normal Form (3NF)**

**Criteria:**

* The database must meet 2NF.
* There should be no transitive dependency (non-key attributes should not depend on other non-key attributes).

**Analysis:**

* The tables avoid storing derived data (e.g., total order amounts, which are calculated from OrderDetails).
* Foreign key relationships (e.g., AddressID in Customers and Staff) are appropriate and don’t violate 3NF.
* Attributes such as CustomerName (in views) are derived during querying rather than stored redundantly.

**4. Develop SQLite Server database system**

**4.1 Database Implementation**

The database was implemented using SQLite, which is a lightweight and efficient database management system suitable for this project. The steps taken include:

1. Obraz zawierający tekst, Czcionka, zrzut ekranu

   Opis wygenerowany automatycznie**Database Creation**:  
   A new SQLite database file was created and named restaurant\_management.db
2. Obraz zawierający tekst, zrzut ekranu, oprogramowanie

   Opis wygenerowany automatycznie**Table Creation**:  
   The following SQL commands were used to create the tables as per the relational schema. Each table includes the necessary primary keys, foreign keys, and constraints to enforce referential integrity.
3. **Referential Integrity**:  
   Relationships between tables were established using foreign key constraints to ensure data integrity. For example:

* AddressID in the Customers table references the AddressID in the Address table.
* MenuID in the MenuItems table references the MenuID in the Menus table.

**4.2 Data Insertion**

Initial sample data was inserted into the database to validate its functionality.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

**4.3 Database Testing**

The database was tested to ensure all relationships and constraints functioned correctly. Examples of tests performed include:

1. **Foreign Key Constraint Test**: Attempting to delete a CustomerID that has existing Orders results in an error, demonstrating that the foreign key constraint is working.
2. **Data Integrity Test**: Trying to insert a duplicate Email into the Customers table results in a constraint violation.
3. **Query Test**: Queries were run to retrieve specific data, such as all orders by a customer or menu items available for order.

**4.4 Database Diagram**

A data diagram was generated from the implemented SQLite database to visualize the relationships and structure of the tables. This diagram matches the previously defined ERD and reflects the implemented system.

Obraz zawierający tekst, diagram, zrzut ekranu, Równolegle

Opis wygenerowany automatycznie

**Part 2: Database design development**

This section focuses on the development and testing of the SQLite database system designed for the restaurant. The aim is to implement advanced database features such as triggers, functions, views, and optimized queries to meet business requirements.

**1.Triggers**

Triggers are used to enforce business rules and automate specific actions within the database. In this system, three triggers have been implemented to meet the operational needs of the restaurant:

**Trigger 1: Automating Order Processing and Payment Recording**

This trigger ensures that whenever a new order is placed, the corresponding order details and payment records are automatically generated.

**Key Functions:**

* Creates a detailed entry in the OrderDetails table for the ordered item, including quantity and price.
* Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

  Opis wygenerowany automatycznieInserts a payment record in the Payments table, capturing the order total and payment method.

**Trigger 2: Real-Time Inventory Management**

This trigger is activated after new order details are added. It automatically updates the stock levels of ingredients used to prepare the ordered menu items.

**Key Functions:**

* Adjusts the StockQuantity in the Ingredients table based on the quantity of ingredients used in the ordered items.
* Prevents overstocking or shortages by maintaining real-time inventory accuracy.

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

**Trigger 3: Financial Tracking Automation**

This trigger is executed whenever a new payment is recorded. It logs the transaction into the FinancialRecords table, providing a detailed record for revenue tracking.

Key Functions:

* Creates an entry in the FinancialRecords table with information about the transaction type, order ID, and payment amount.
* **Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

  Opis wygenerowany automatycznie**Streamlines revenue reporting by ensuring all payments are accounted for.

**2.** **Reusable Function Development**

**3.Views**

**1. Orders Summary View**

**Obraz zawierający tekst, zrzut ekranu, wyświetlacz, Czcionka

Opis wygenerowany automatycznie**The OrdersSummary view provides a detailed summary of all orders, including customer and staff details, order date, total amount, and payment method.

**2. Ingredient Stock Levels View**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**The IngredientStockLevels view shows the stock levels of all ingredients, including their unit of measure and associated supplier details.

**3. Menu Items with Ingredients View**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznieThe MenuItemsWithIngredients view lists all menu items alongside their respective ingredients, quantities, and menu names.

**4. Financial Summary View**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**The FinancialSummary view provides a concise overview of revenue and transactions grouped by record type.

**5. Financial Details View**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie**The FinancialDetails view provides a detailed record of all financial transactions, sorted by date.

**4. Queries Meeting Business Requirements**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie1. Query with a Message and JOIN**

**Output**

**Input**

**2. Query with GROUP BY and HAVING**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

**Input**

**Output**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie**

**3. Query Using SQLite Functions**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie**

**Input**

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

**Output**

**4. Query with a Message and JOIN**

**Input**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

**Output**

**5. Query that combine multiply features**

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

**Input**

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

**Output**

**5. Testing and reviewing the database.** To ensure that the database functions as intended, various tests were conducted to verify the integrity of the schema, relationships, and additional features such as triggers and views. The testing process included the following steps:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznieObraz zawierający tekst, Czcionka, linia, zrzut ekranu

Opis wygenerowany automatycznie**1.Trigger validation**

**Output**

**Input**

**Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie2.View validation**

**Output**

**Input**

Obraz zawierający tekst, Czcionka, zrzut ekranu, design

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie**3. Error Handling**

**Output**

**Input**

**4. Integrity Testing**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie**

**Input**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie**

**Output**

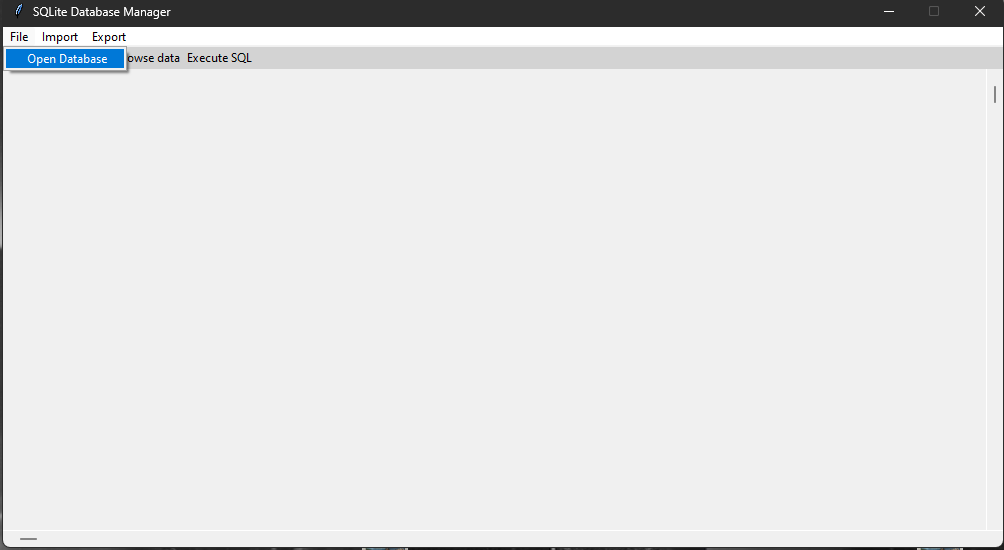
**Part 3: Programming for Database and test**

This section provides the user with a Python Tkinter software application for managing the database. It allows user to view, insert, update, delete data in the database and create reports.

**1.Communitation with SQLite by Python GUI**

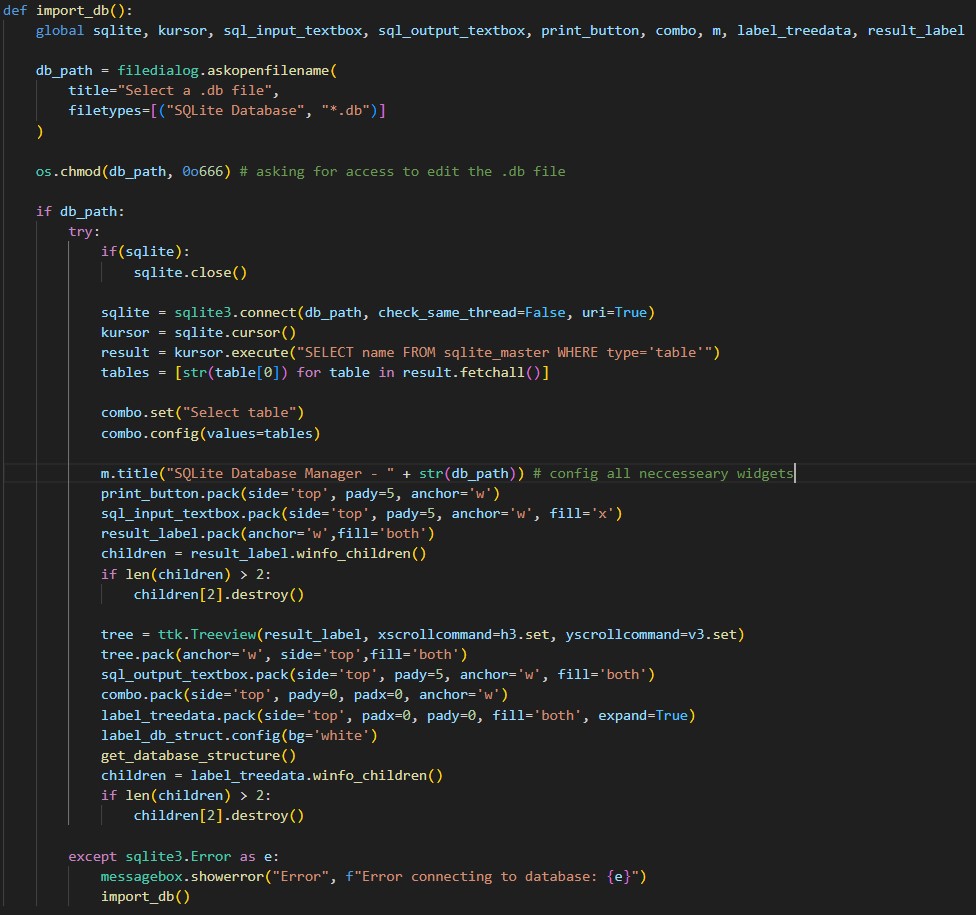
After starting the application, the user sees a menu bar and a navigation bar with various options like:

**1. Opening database from SQLite file**

****

The “Open Database” command under “File” cascade allows user to open the SQLite file with database





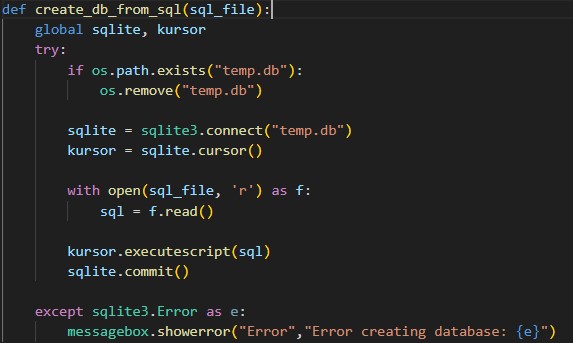
**2. Import database from SQL or XML File**

****

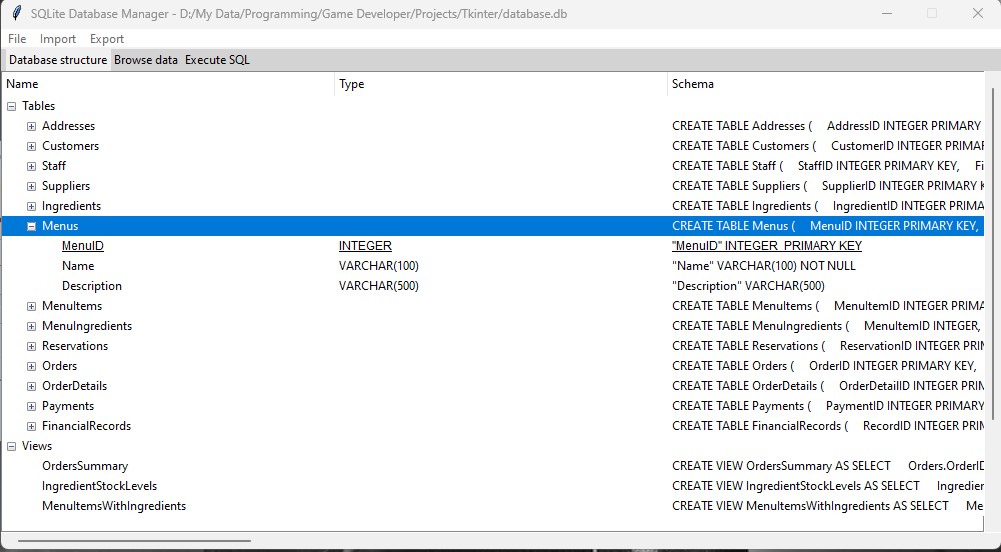
Commands under “Import” cascade allow user to Import database either from SQL file or XML file.

Note: If the database is imported from a non-SQLite file, the program creates a temporary SQLite database in the directory. If the user needs to preserve the changes to the database, it is necessary to export the database.

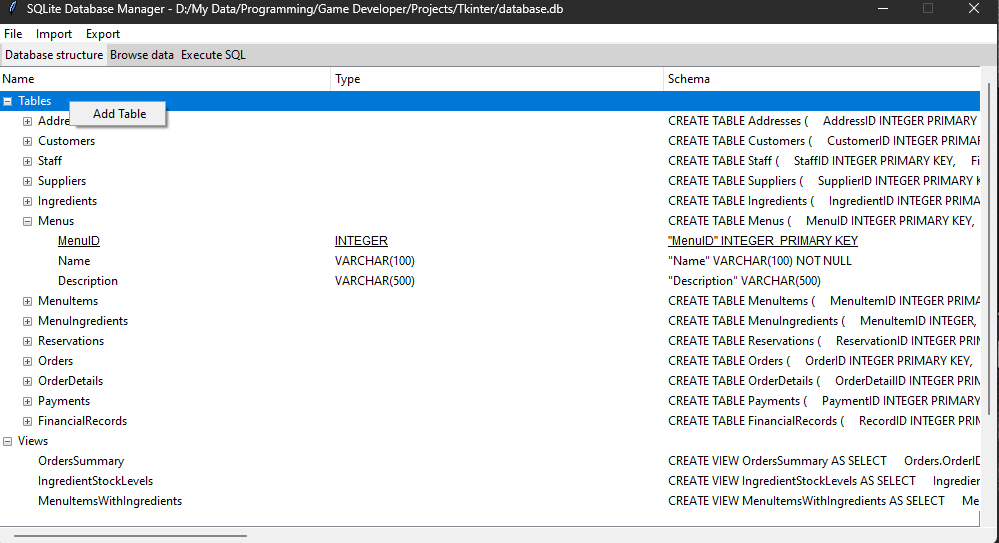


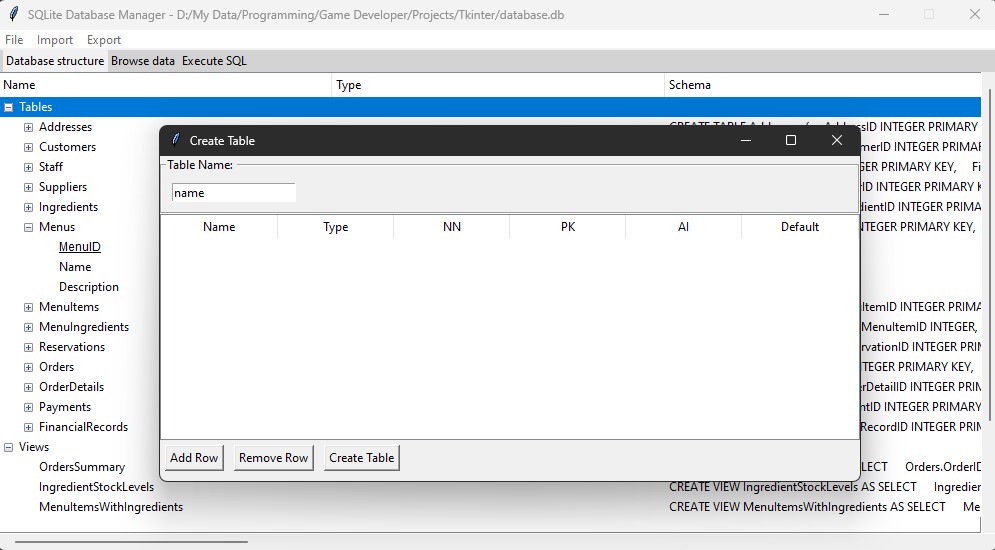


1. **Preview database structure**

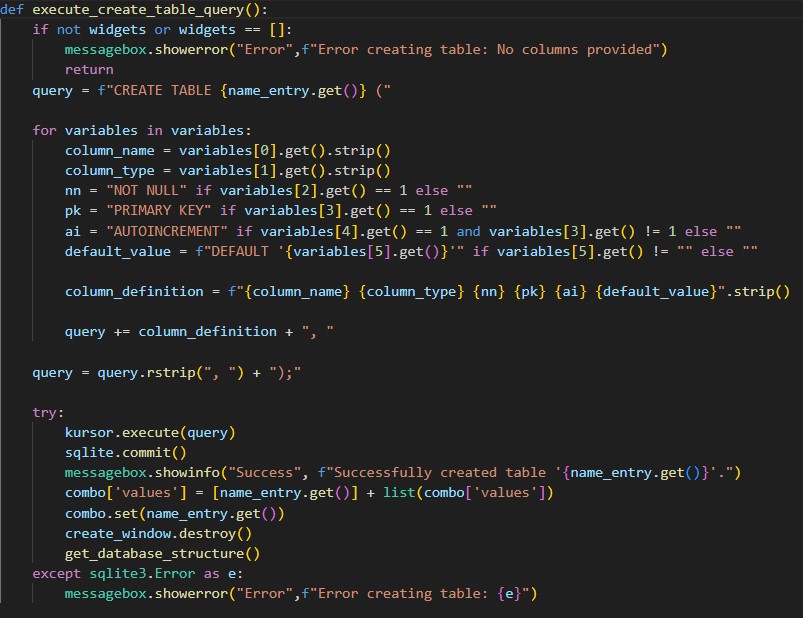
After opening or importing the database, the user has access to the database structure. This is useful because it provides knowledge about all tables, data types, triggers, views in one place, which can help with modifying data.

Additionally in this view, the user can insert a new table into the database by right-clicking on the “tables” node.



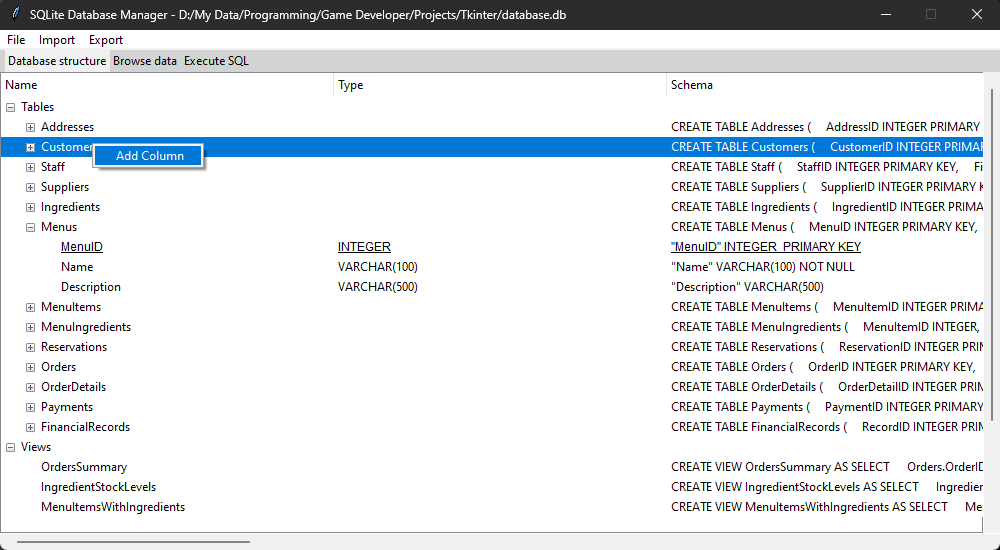


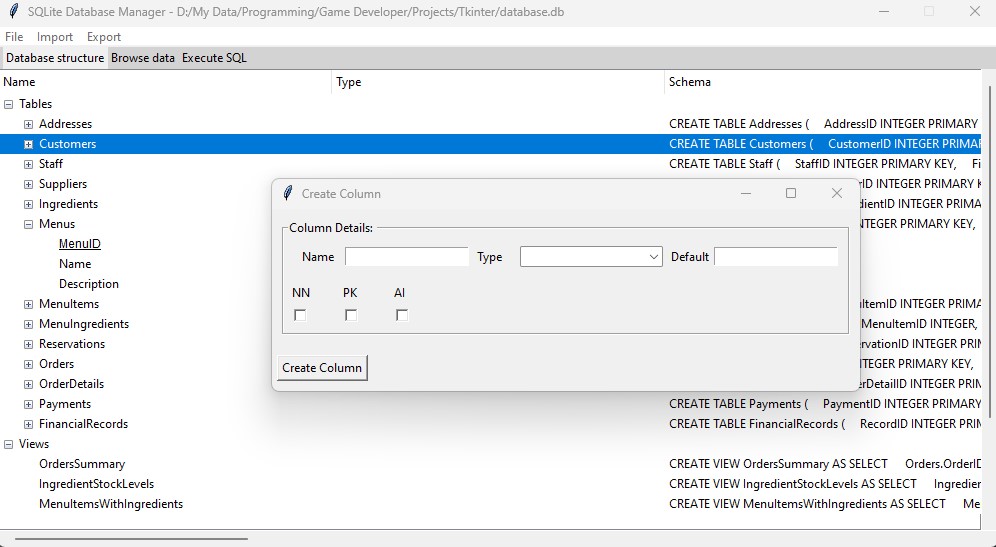
The user must provide a table name, add columns, name them and assign constraints attributes.



This is the code that interprets user-entered data and executes the query that creates a new table.

User is also given the option to add a column to an existing table:

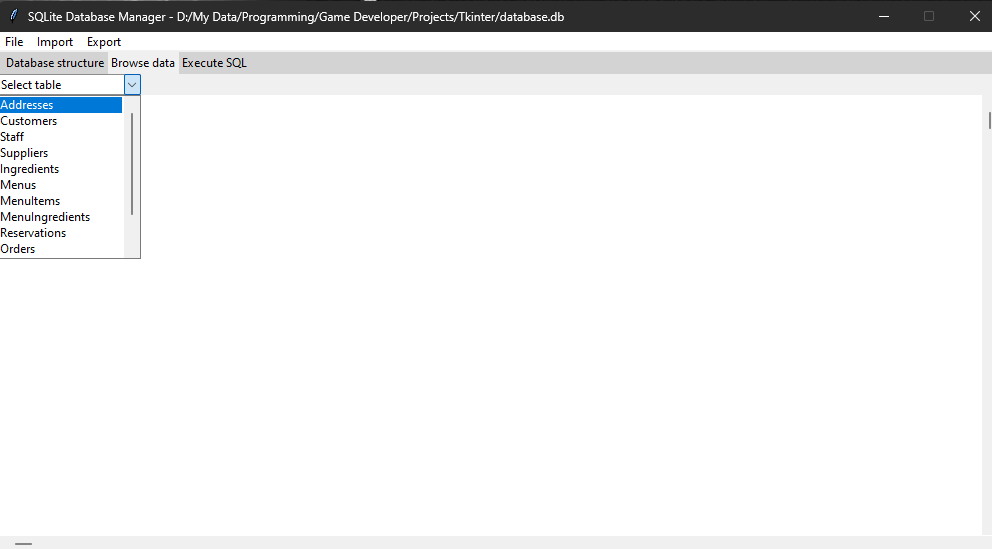


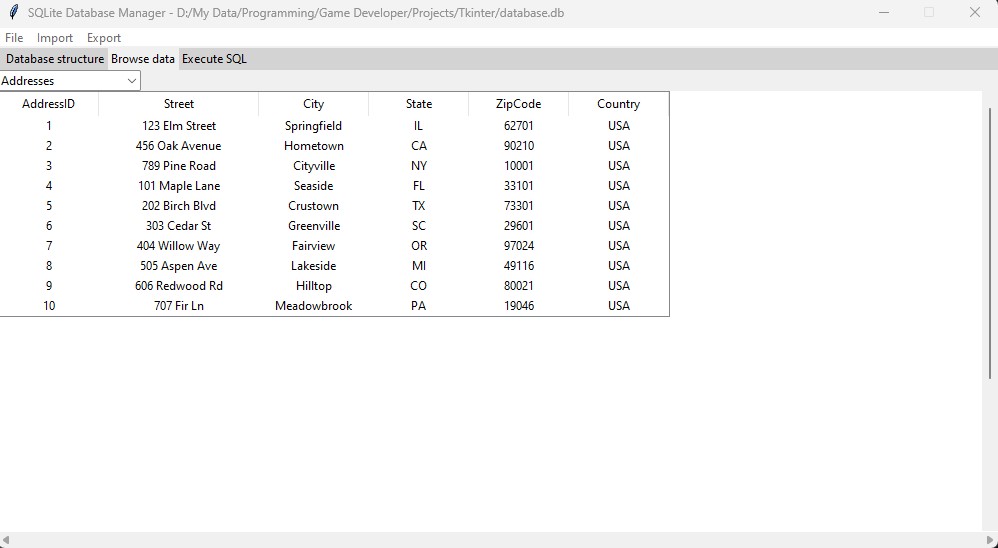
****

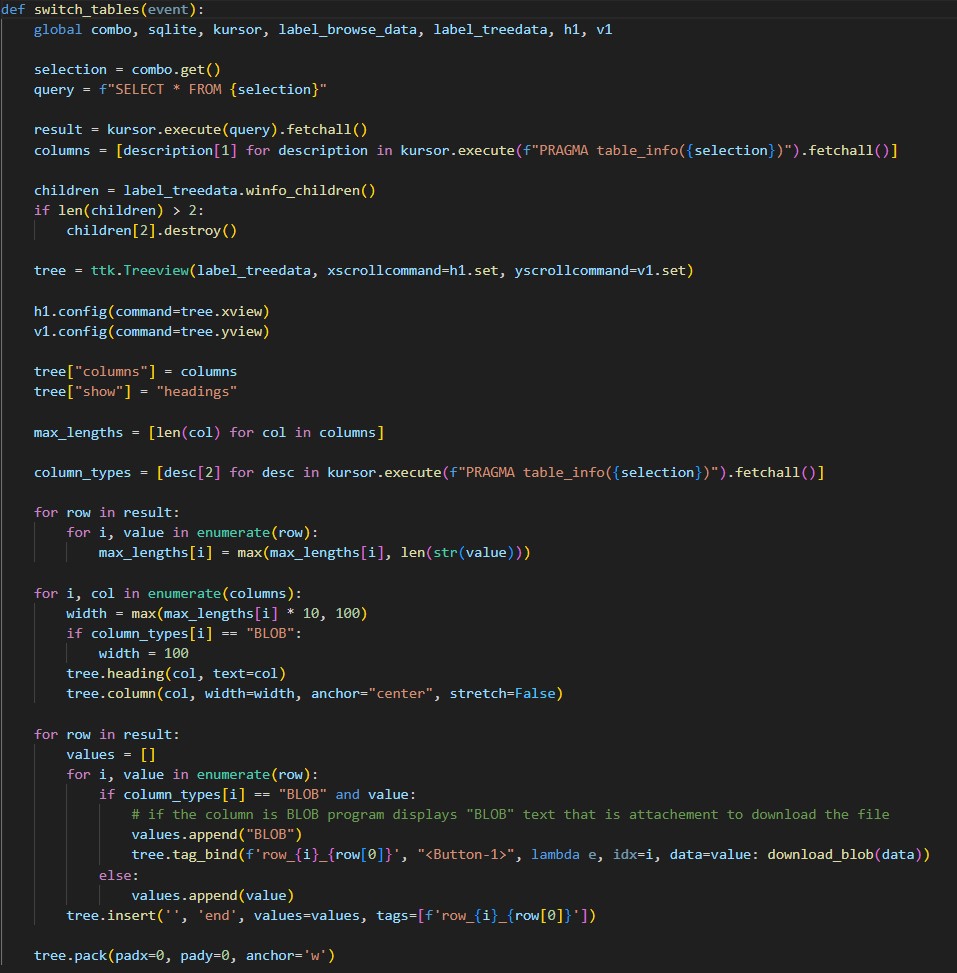
****

1. **Browse all data in the database**

The user can also see all the tables in the database. By selecting one in the combobox, he has a preview of the table:

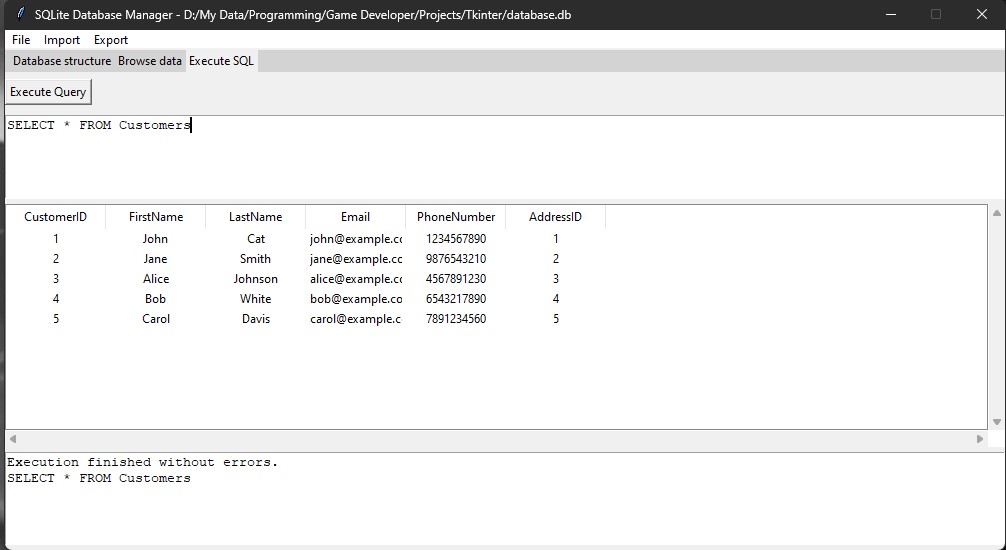
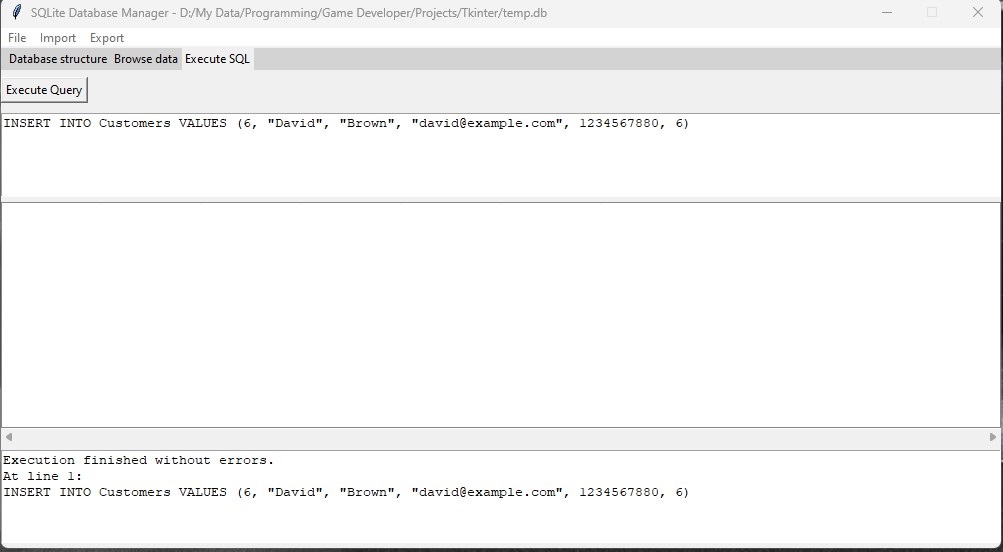


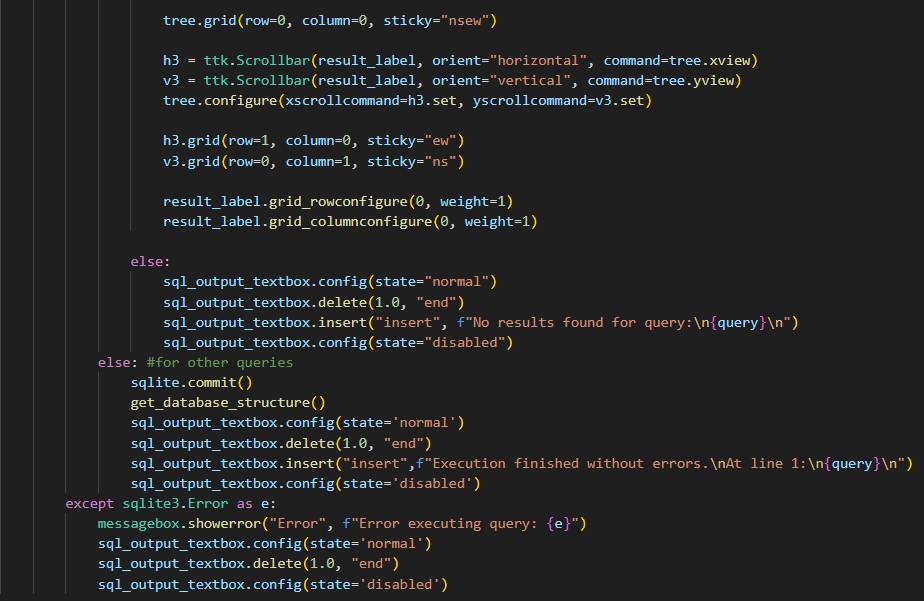




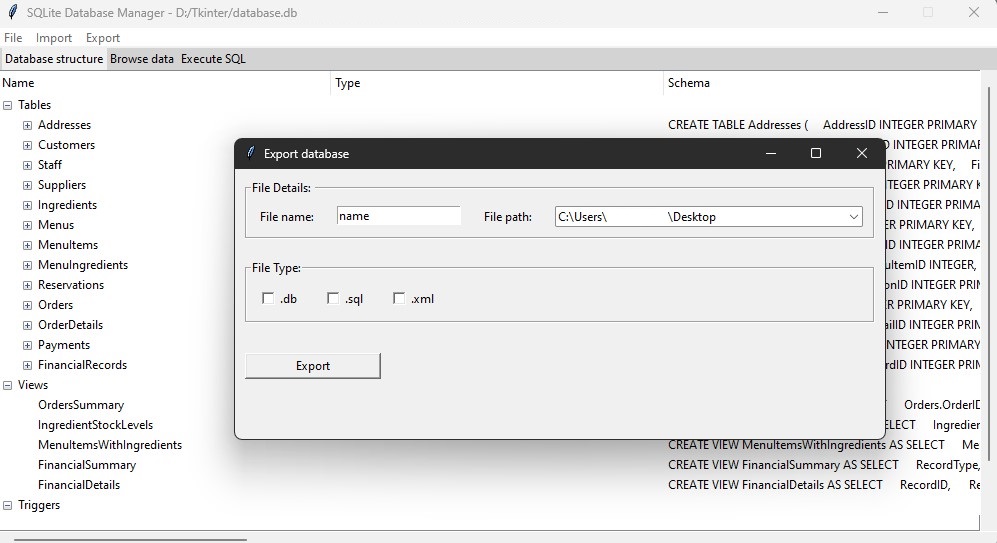
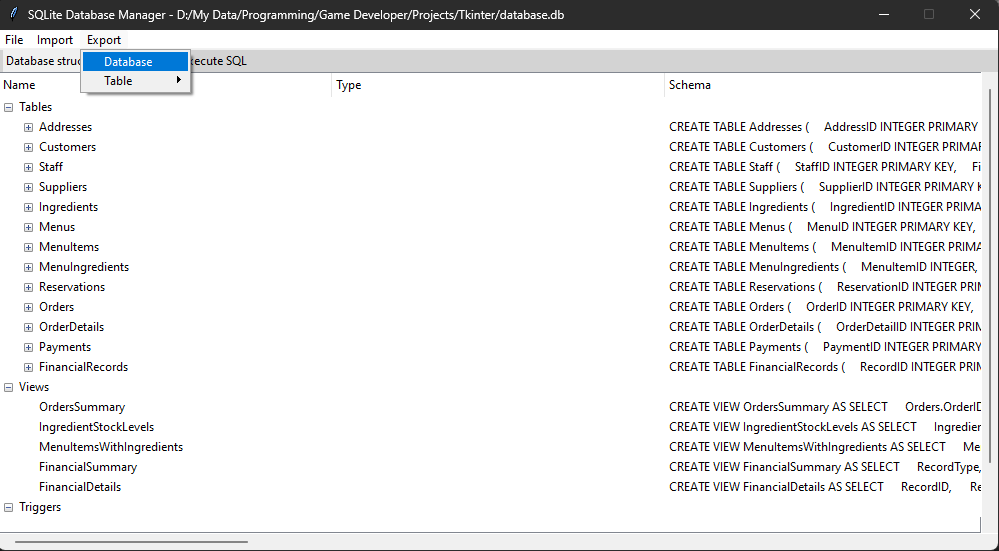
1. **Executing SQL queries**

In the Execute SQL tab, the user can enter SQL queries and execute them. When the query syntax is correct, the operations are executed and a positive message is returned in the text box. If the query (e.g. SELECT) returns a result, a table with the appropriate records will be visible in the field between the input textbox and the output textbox

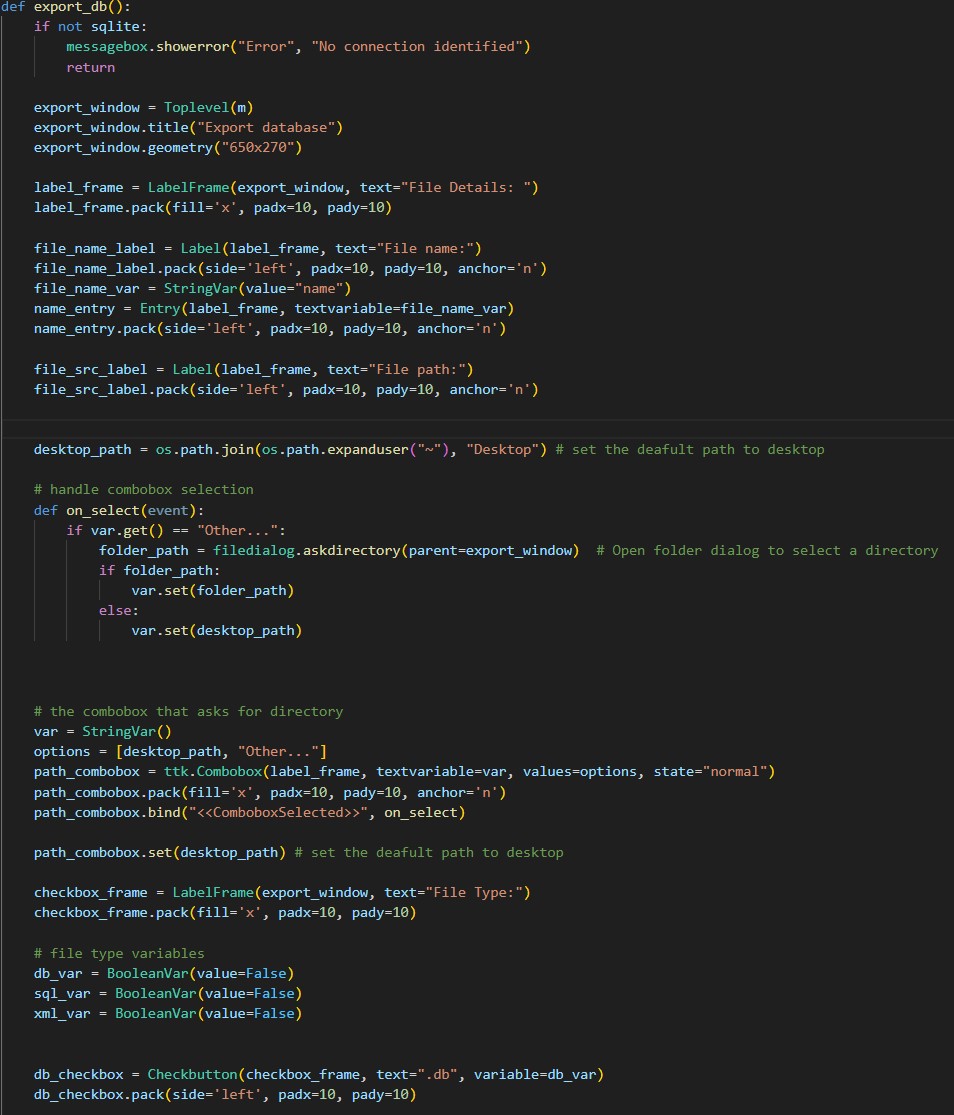


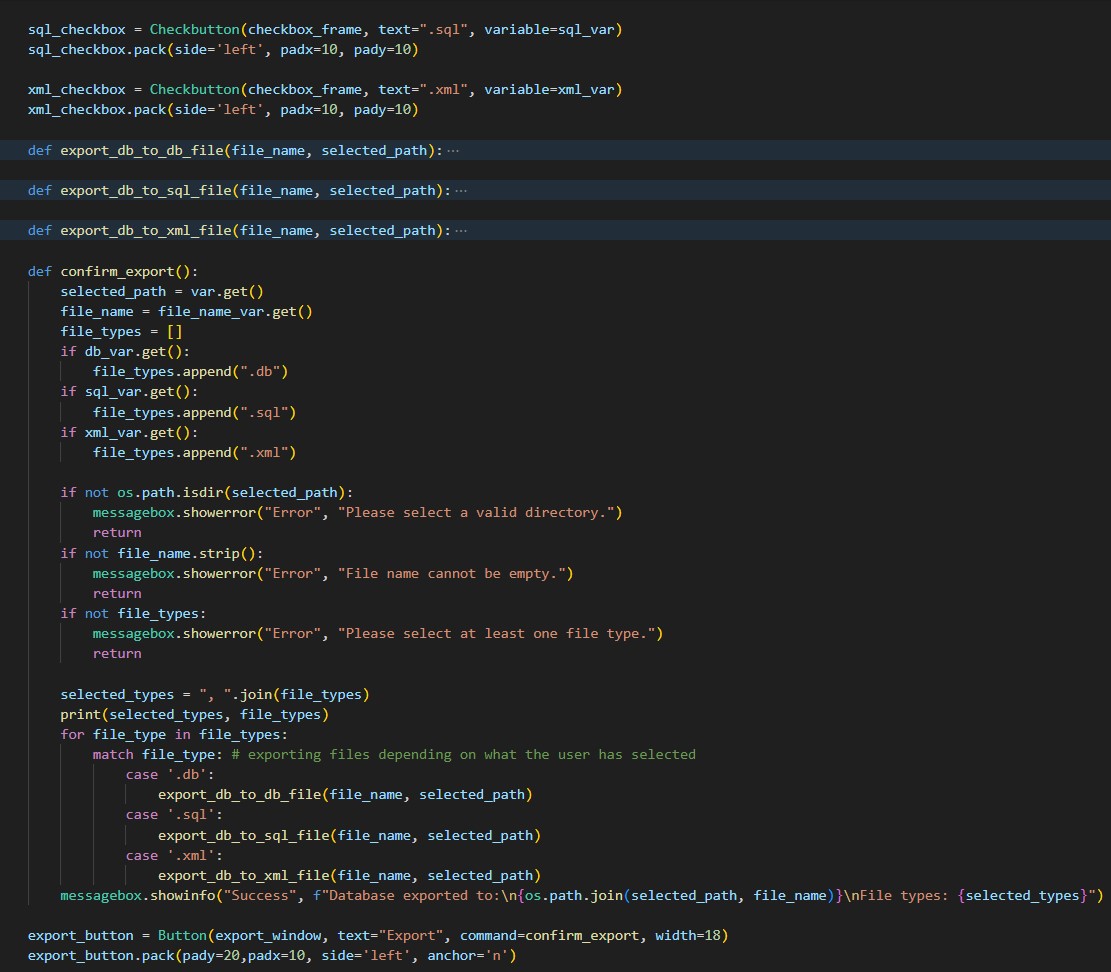


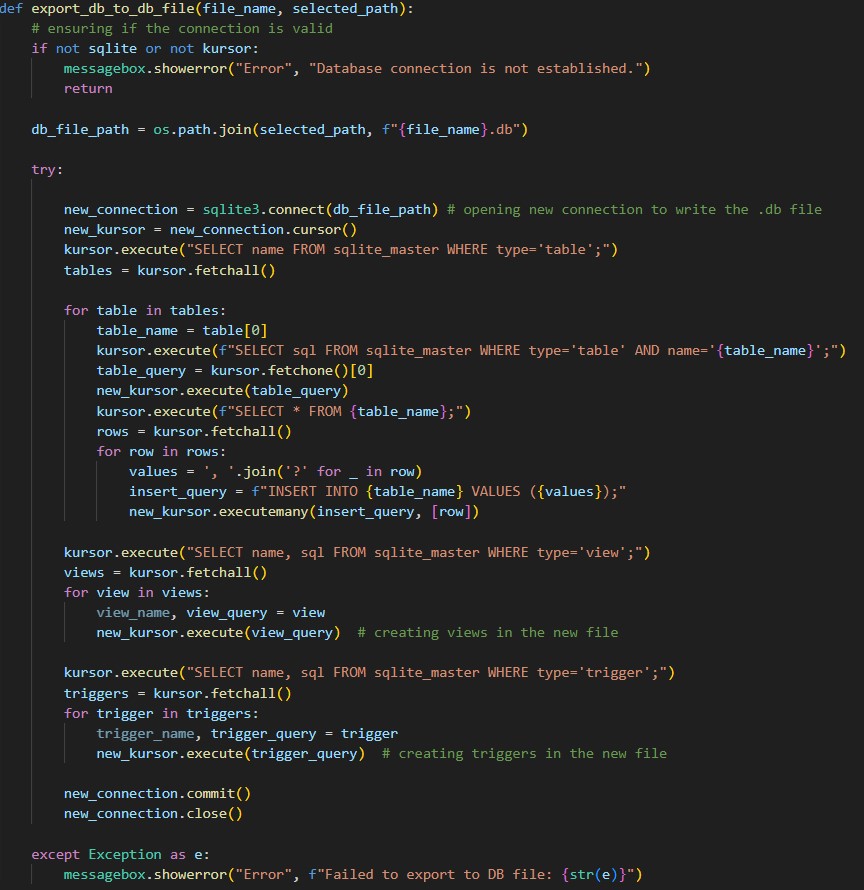
1. **Export database to SQLite, SQL or XML file:**

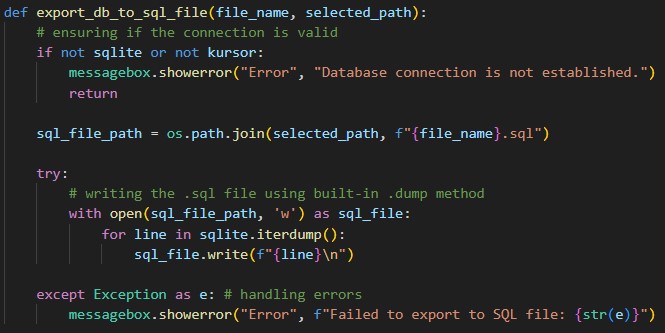
By clicking “Database” in the “Export” cascade user can export database to either SQLite, SQL or XML file or multiple at once. 

The user must provide the file name, path where the file or files are to be saved and the extension. After clicking the export button, a message will be displayed whether the export was successful and if so, the file or files with the database will be saved.





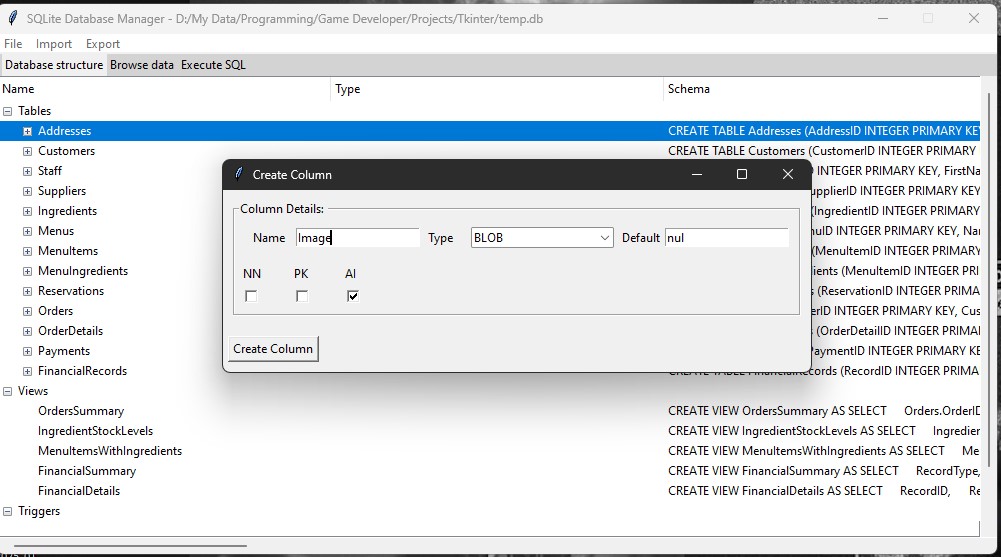


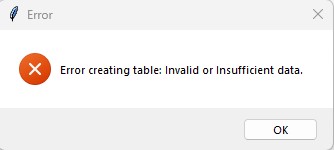
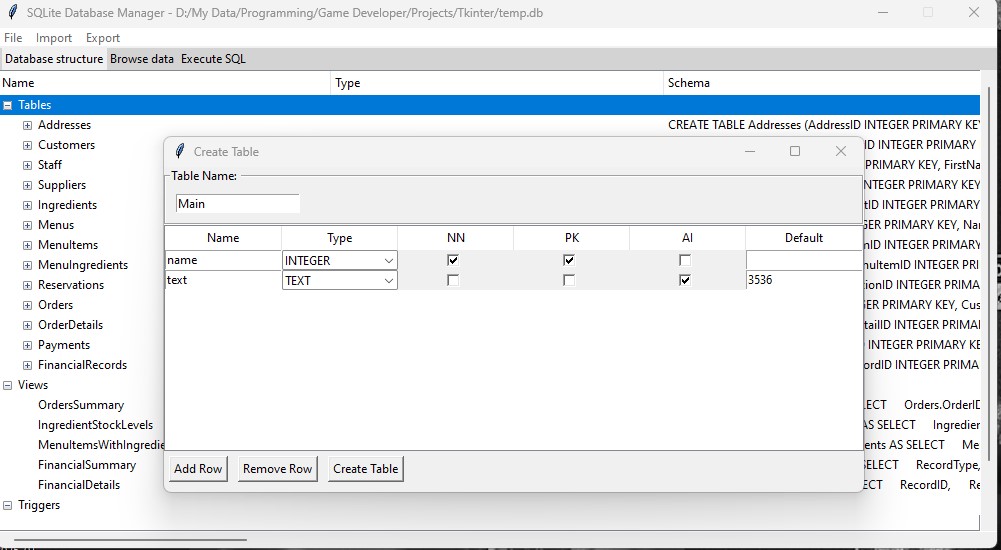
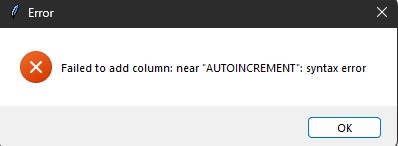


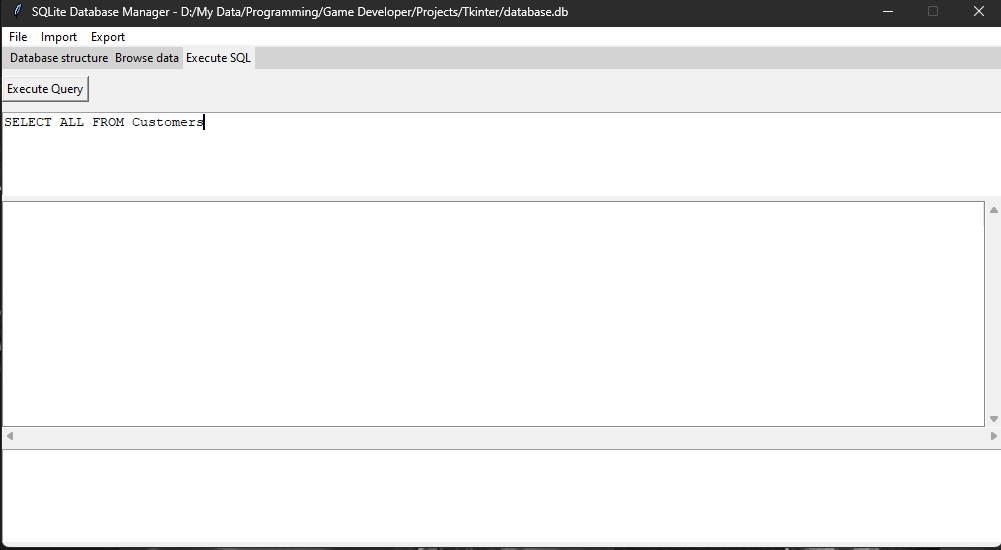
**2. Validation or verification of input data**

As mentioned before, if the user enters incorrect data for creating a table or column, data that conflicts with existing data in the database, or enters a query with incorrect syntax, the program will inform the user about the error and will not perform the operation until the data is correct.

Examples:





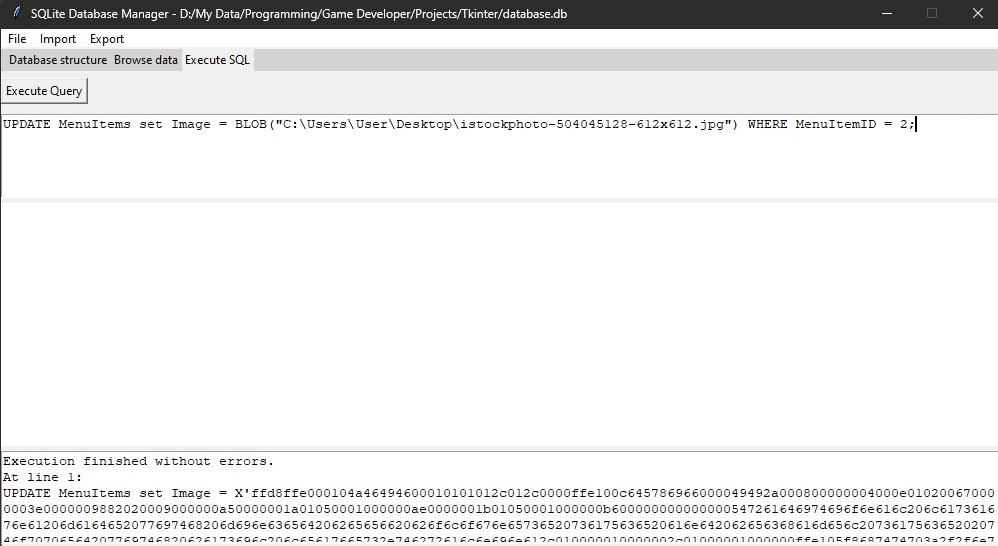


**3. Storing images in database using BLOB data type**

The program can write to the database, read from the database, and save BLOB files.

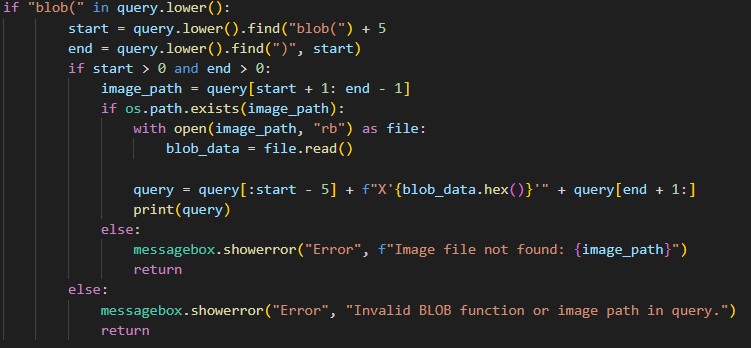
When entering an UPDATE query for a column with the BLOB data type, the new value assigned by the user should use the BLOB function. Its specifically designed for program usability and simplicity, and takes the file path as an argument.

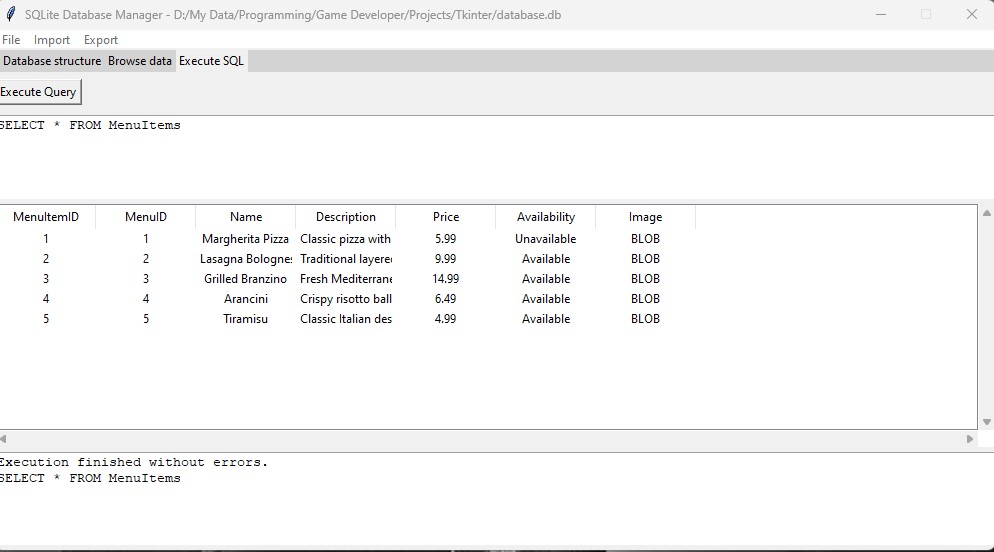
Here is an example of correct usage:



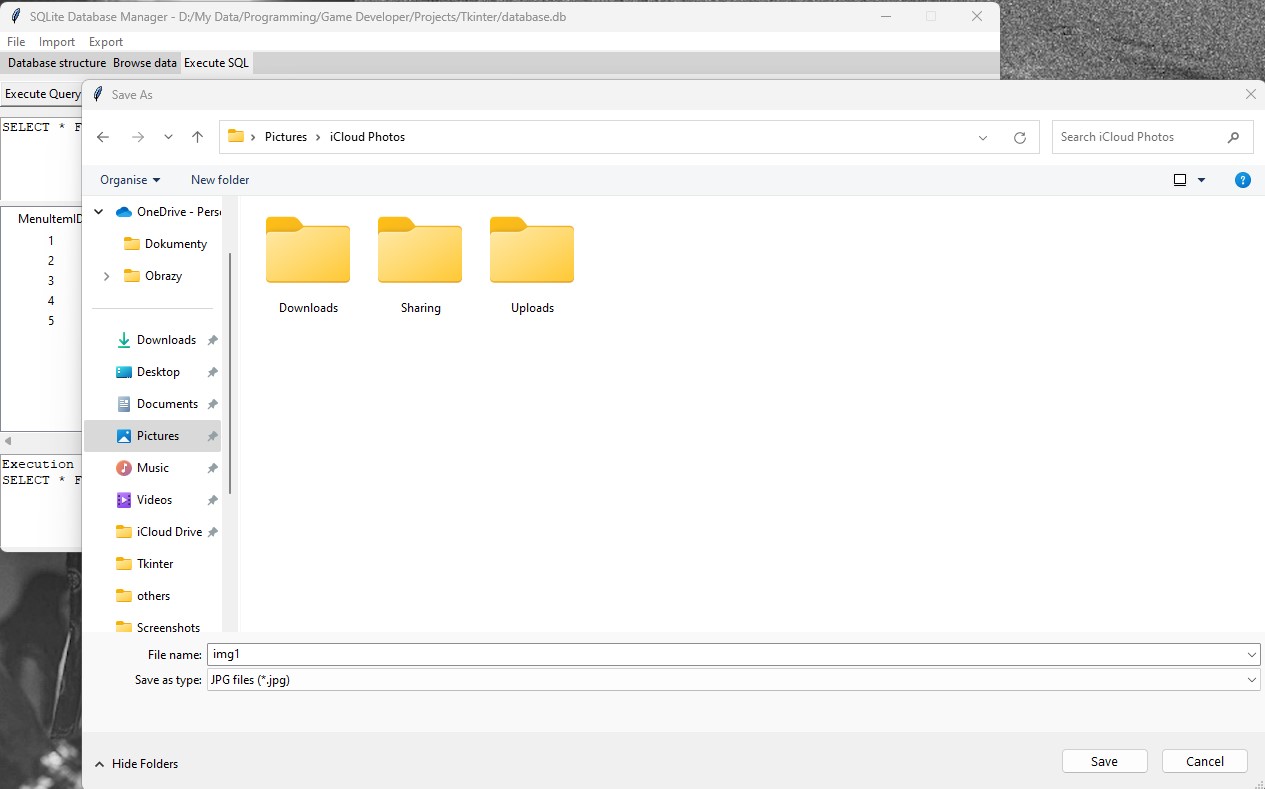
In this example, in the output textbox, the user can see the actual query that was compiled by the program. The user inserted a blob object into the Image column.

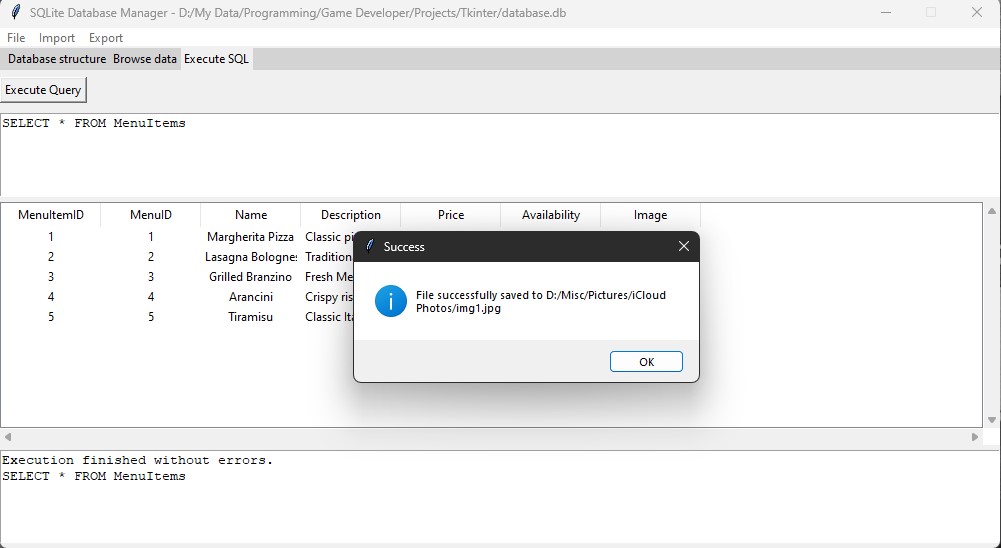
Here is the code that is inserting the BLOB file to the database:

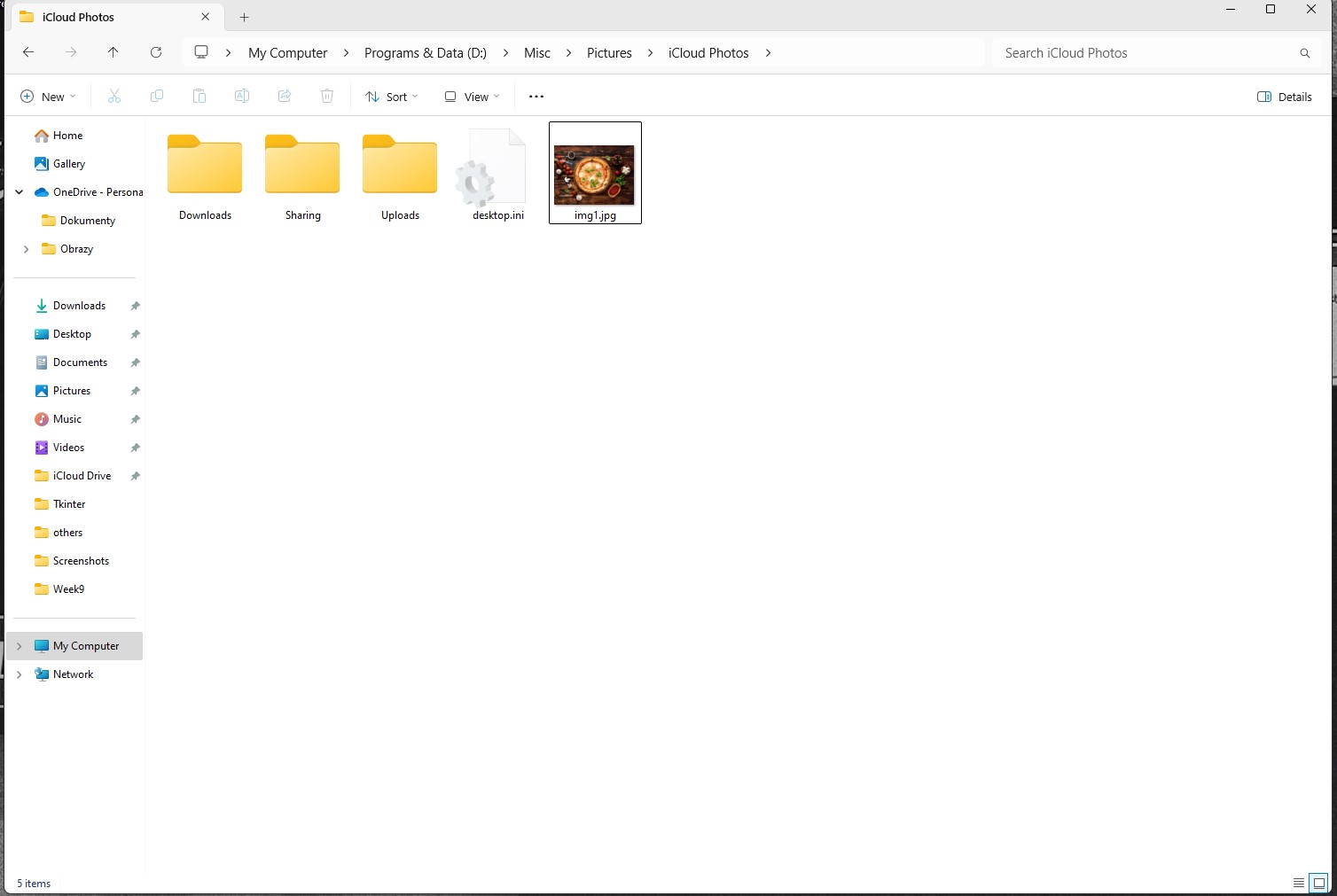
In execute\_query function:  
  
When a user would like to read a blob file, SELECT query

that has this data type can be executed:

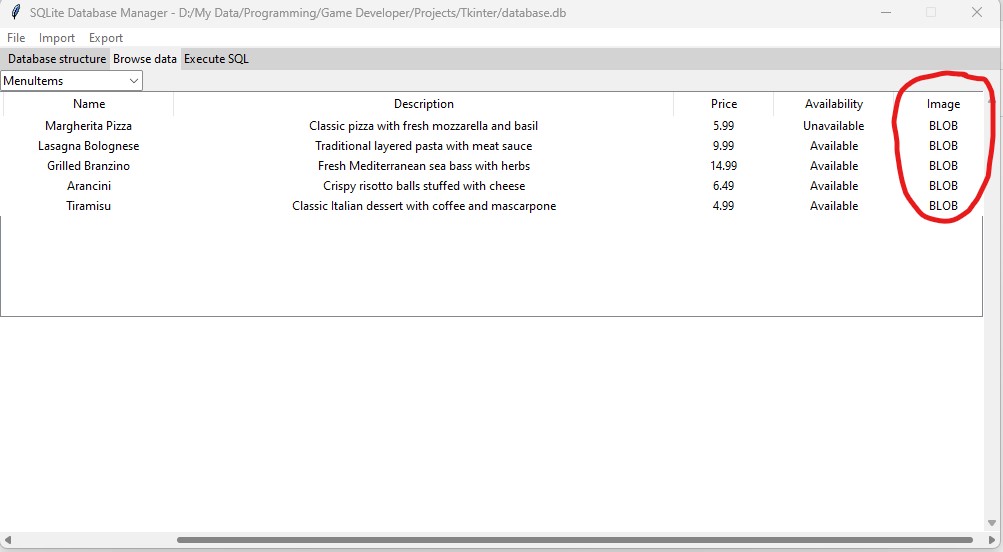
In the data place there is a BLOB placeholder which, when clicked with left mouse button, downloads the blob file that is saved in the given cell.

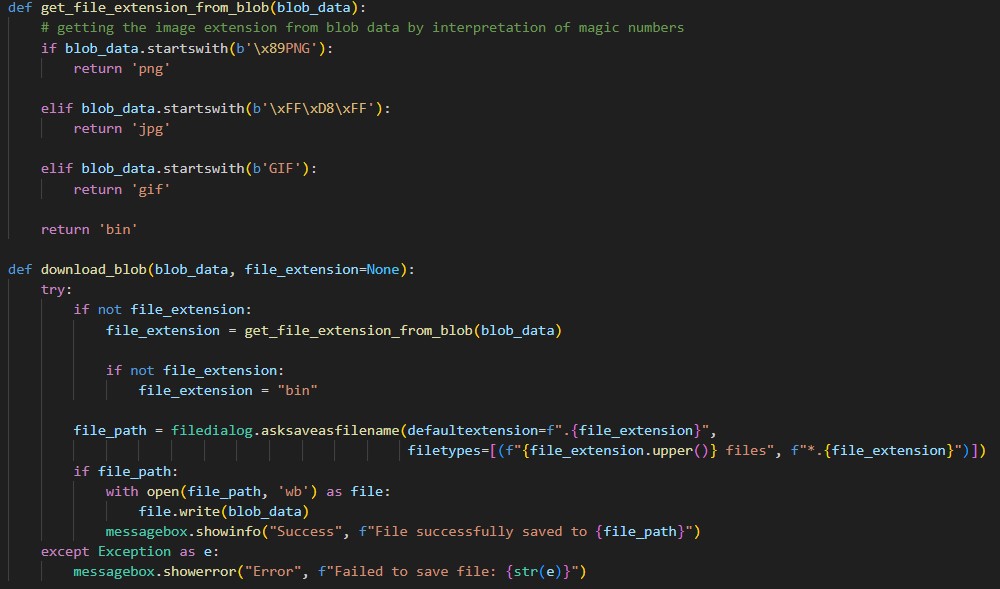






Same goes for tables in “Browse data”:

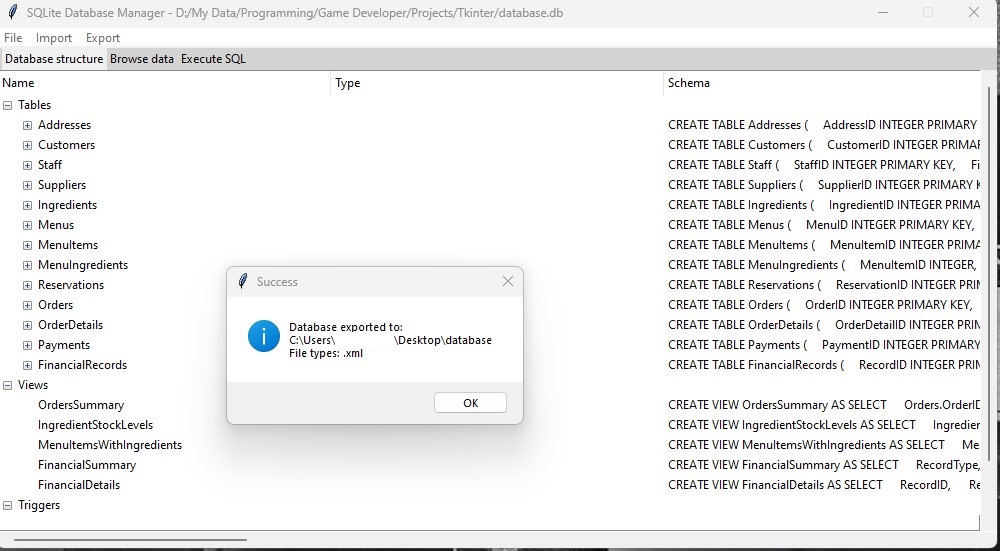




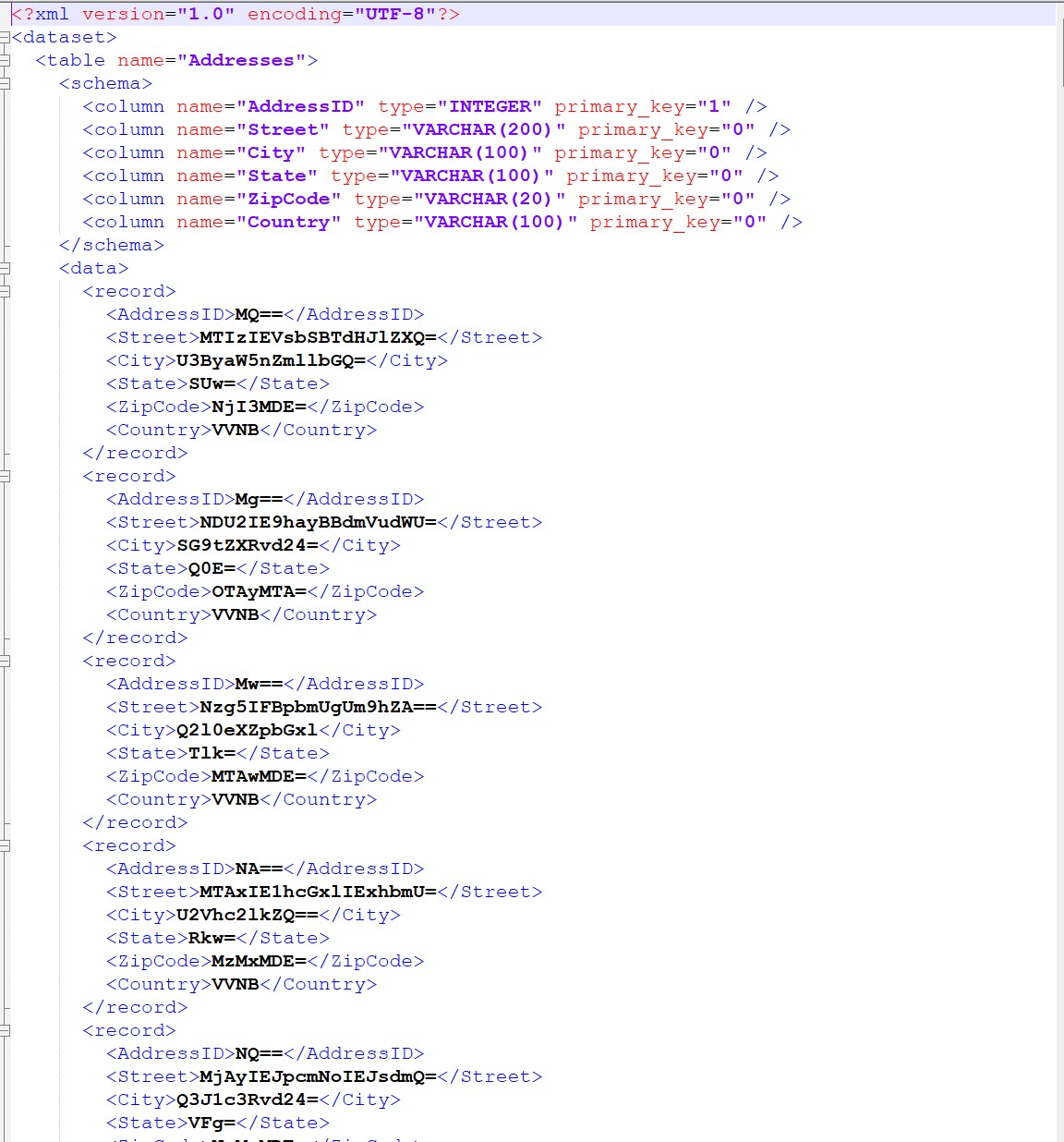
**4. XML Development**

The application supports exporting the database in XML format importing the database in this format which is limited only to XML databases that have been exported by this application.





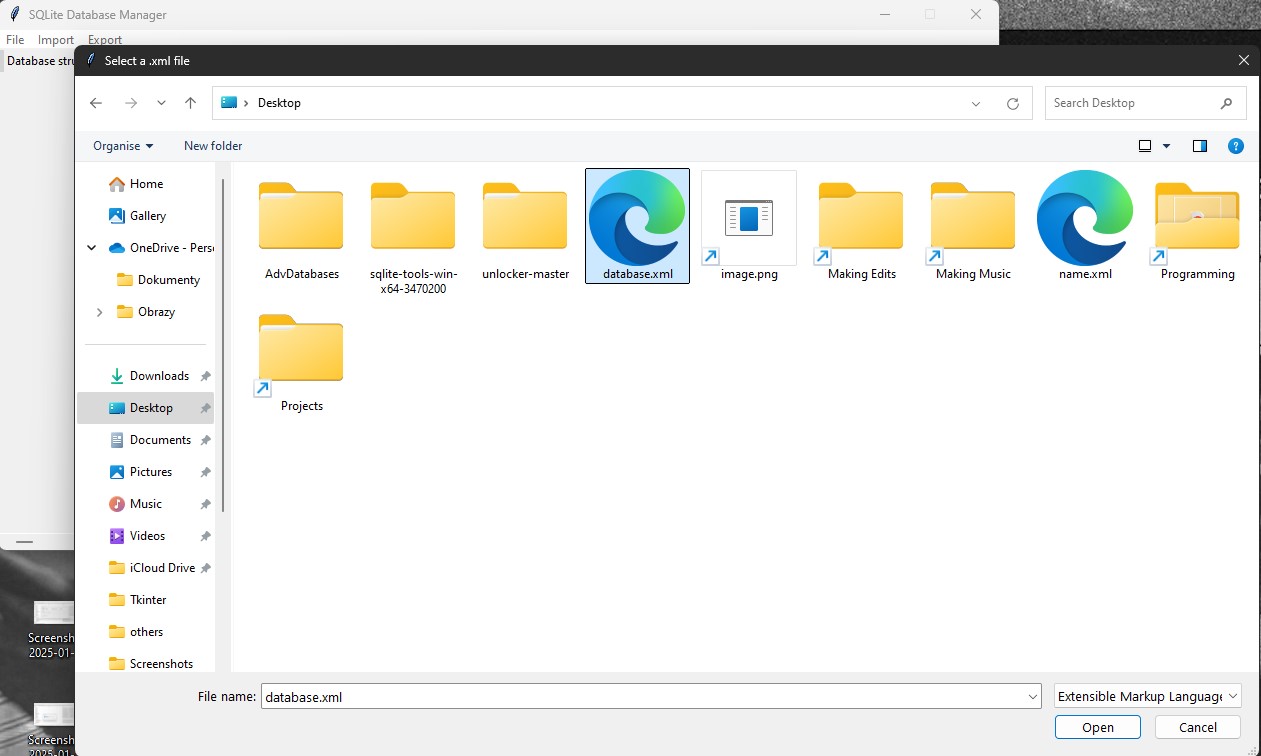
In database.xml:

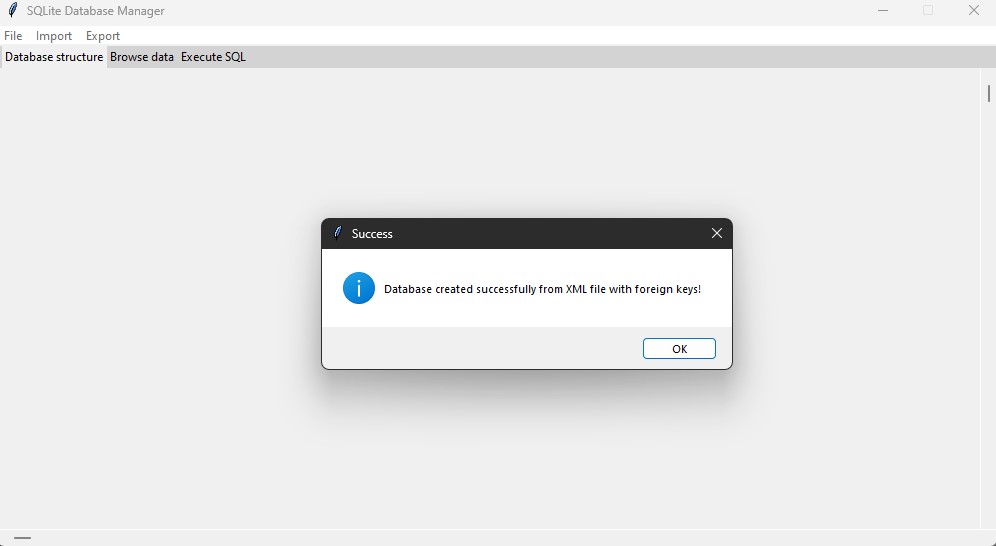


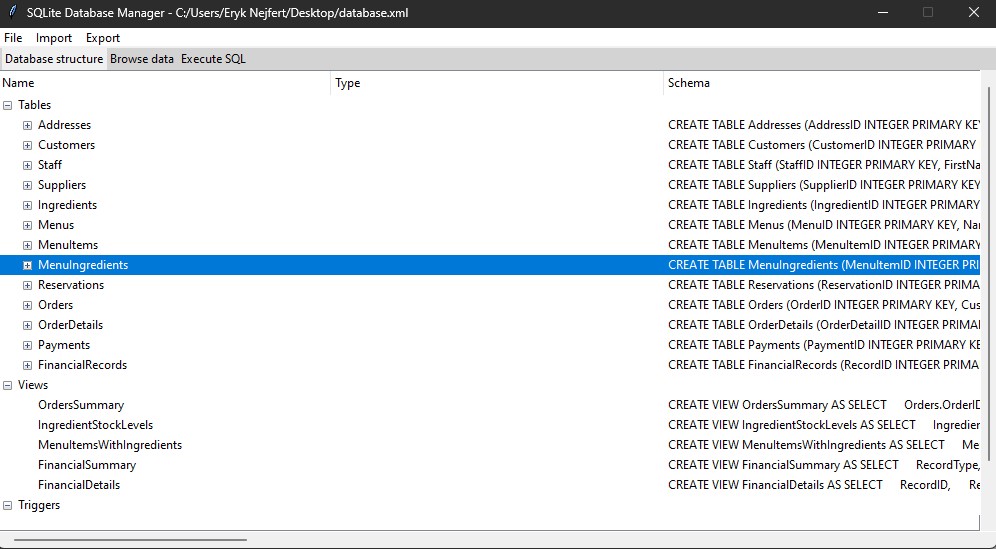


When the user would like to import the database.xml:



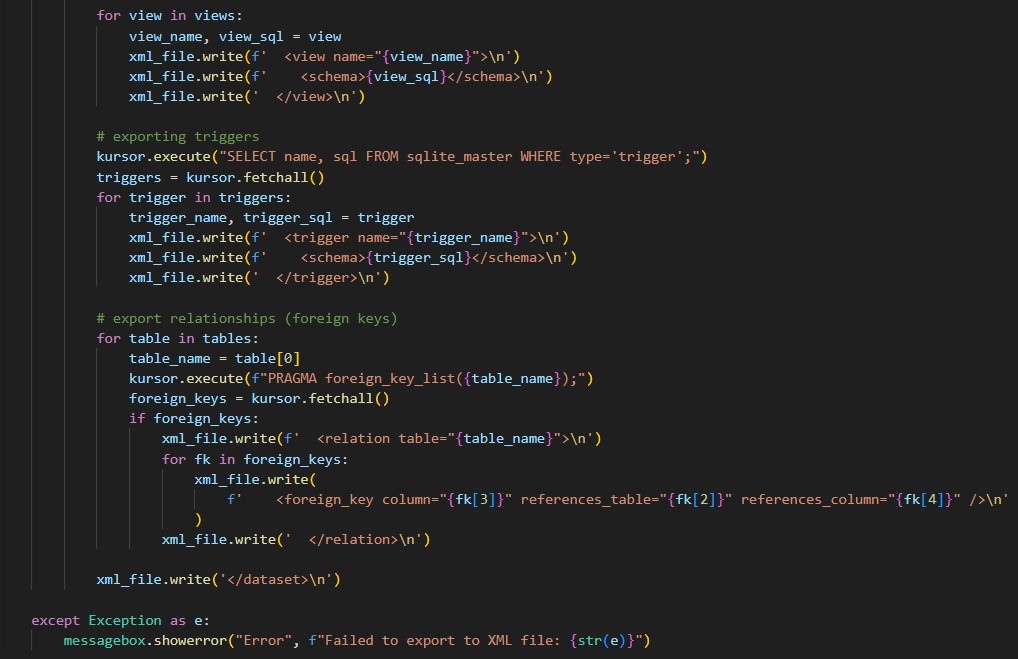




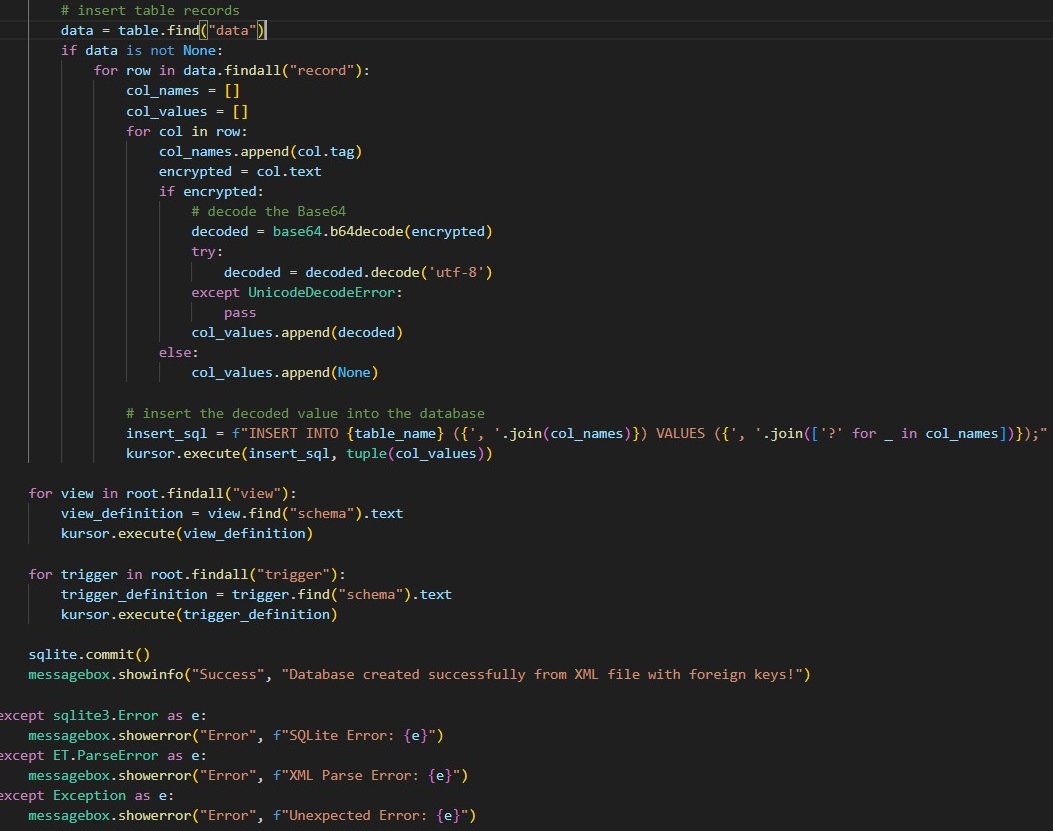


The program encodes the data in Base64 and then writes data in new XML file, and decodes data from XML file and reads data, then writes them in temporary SQLite database:









**5. Innovative features in the application**

The application has many innovative functions and solutions that improve user comfort, such as inserting images into the database, adding tables and columns from the database structure view and the ability to download images from the database. It is very easy to use and has unusual but working solutions.

In addition, the application supports all the most popular data saving methods (SQLite, SQL, XML and even CSV which allows for quick export of tables to a csv file and then e.g. importing them to excel) and efficiently operates between them.

Exporting table to CSV file:

