Database Introductory Course – Final Project

Summary

I - SQLite Essay

Write an essay on one of the following topics. Your essay should demonstrate your understanding of the chosen topic, its relevance in database systems, and practical applications. Use references and cite them using IEEE citation style.

Introduction

For many years, the world became more and more digital. In the past, information was stored in physical books and archives, but the digital transformation has considerably facilitated its storage, search, and accessibility, which eliminated physical constraints. To this end, the concept of « database » emerged. It is a centralized system for storing a large volume of data, which led to the creation of database management systems (DBMS) designed to efficiently create, manage, and access this data. SQLite, one of the most popular DBMS, is currently ranked 11th in the DB-Engines popularity ranking [1]. This essay will explore the nature of SQLite, the reasons why it is useful, and its various applications.

The nature of SQLite : A Serverless Approach

SQLite is a software library written in C that provides a relational and transactional database engine. This means it is a system where data is organized into tables, and these tables can be linked together using primary and foreign keys. Its name is a contraction of « SQL » and « Lite » reminding its simplicity and lightness, a principle that has guided its development since its creation in the early 2000s by D. Richard.

As GeeksForGeeks states, « SQLite is a highly efficient, serverless, and self-contained SQL database engine that stands out for its simplicity and ease of integration. » [2]. This serverless architecture is one of the main key features of this DBMS. Unlike traditional client-server DBMSs like MySQL or PostgreSQL, SQLite does not require an external server to operate. Instead, the data is stored directly in a single file on the host's filesystem. This design choice simplifies the deployment and management process immensely, because it is no longer necessary to have network connections and to configure a server. The database file can simply be copied, moved, or deleted like any other file, which makes SQLite one of the best solutions for most of the software applications.

Portability and Performance: The Core Advantages

The design of SQLite offers big advantages without regarding its serverless nature, especially in terms of portability and efficiency. Because the entire database is contained within a single file, it can be easily attached to applications without complex setup procedures. This makes it an ideal solution for developing cross-platform applications that need to work seamlessly on Linux, macOS, and Windows. A developer can simply include the SQLite library and a database file, and the application is ready to manage its data.

Furthermore, its lightweight nature means it does not use a lot of memory and disk resources, allowing it to function effectively on IoT devices, as described in DevOPSSchool article [3]. For these small systems, SQLite provides the power of a full SQL database without the the use of a large server, enabling local data storage and analysis. The combination of portability and low resource requirements makes it a powerful and polyvalent tool for a vast range of projects, from simple desktop utilities to complex mobile apps.

Data Integrity and Reliability

Even though SQLite is a lightweight DBMS, it is still a reliable one. Actually, SQLite adhere to the ACID properties : Atomicity, Consistency, Isolation, Durability [4].

- Atomicity means that transactions are binary : it either success or fails. If the transaction success, we say that it « commited ». In the other hand, if a transaction fails, we say that it « aborted ».
- Consistency ensures that the database must remain in a valid state before and after a transaction, rules are defined to constraint the transactions. Concretely, if a transaction violates a rule, the system performs a roll-back (or abortion) to prevent data corruption.
- Isolation mens that each transaction is independant and doesn't affect each other : changes made by one transaction is not visible unless it is commited.
- Durability guarantees that once a transaction has been committed, its changes are permanently saved and will not be lost. Even in there is a system failure or a power loss, the data remains intact because it is not stored in volatile memory.

This balance of simplicity and reliability is what makes SQLite apart from other DBMS and allows it to be so used.

A Wide Application Range

The unique properties of SQLite led to its adoption in a lot of modern technologies. As a lightweight and efficient database, it is a used in many mobile applications on different systems such as Android and iOS, where it is used to store local application data, user settings, and cached information. This allows applications to function even when the phone is offline. In addition, SQLite is a vital component of many desktop applications, including popular web browsers like Google Chrome and Mozilla Firefox, which use it to manage browsing history, cookies, and bookmarks [3]. Its simplicity also makes it great for small web projects or for developers building apps that needs to save data. To finish, as precedently mentionned, its minimal resource requirements make it perfect for IoT devices, where it can manage data logs and device configurations.

Conclusion

In conclusion, SQLite's serverless architecture, simplicity, and remarkable portability have made it a truly indispensable Database Management System. Its unique ability to provide a unique transactional database engine without the need for a dedicated server or configuration has secured its place as a main component of modern software. The adoption of mobile-technologies like smart watches or rings will make the use of SQLite necessary for the next years, and that is the essence of this DBMS.

References

[1] DB-Engines, "DB-Engines Ranking," DB-Engines, [Online]. Available: https://db-engines.com/en/ranking. [Accessed: 12 September 2025].

[2] GeeksForGeeks, "Introduction to SQLite," GeeksForGeeks, [Online]. Available: https://www.geeksforgeeks.org/sql/introduction-to-sqlite/. [Accessed: 12 September 2025].

[3] DevOPSSchool, "What is SQLite and use cases of SQLite," DevOPSSchool, [Online]. Available: https://www.devopsschool.com/blog/what-is-sqlite-and-use-cases-of-sqlite/. [Accessed: 12 September 2025].

[4] GeeksForGeeks, « Acid Properties in DBMS », GeeksForGeeks, [Online]. Available : https://www.geeksforgeeks.org/dbms/acid-properties-in-dbms/ . [Accessed : 12 September 2025].

II – UML Diagram

Create an Entity-Relationship Diagram (ERD) or a UML Class Diagram of a database model for an online store. Your diagram should include:

- Entities (e.g., Products, Customers, Orders, Payments)
- Attributes for each entity
- Relationships between entities (e.g., one-to-many, many-to-many)
- Primary and foreign keys where appropriate

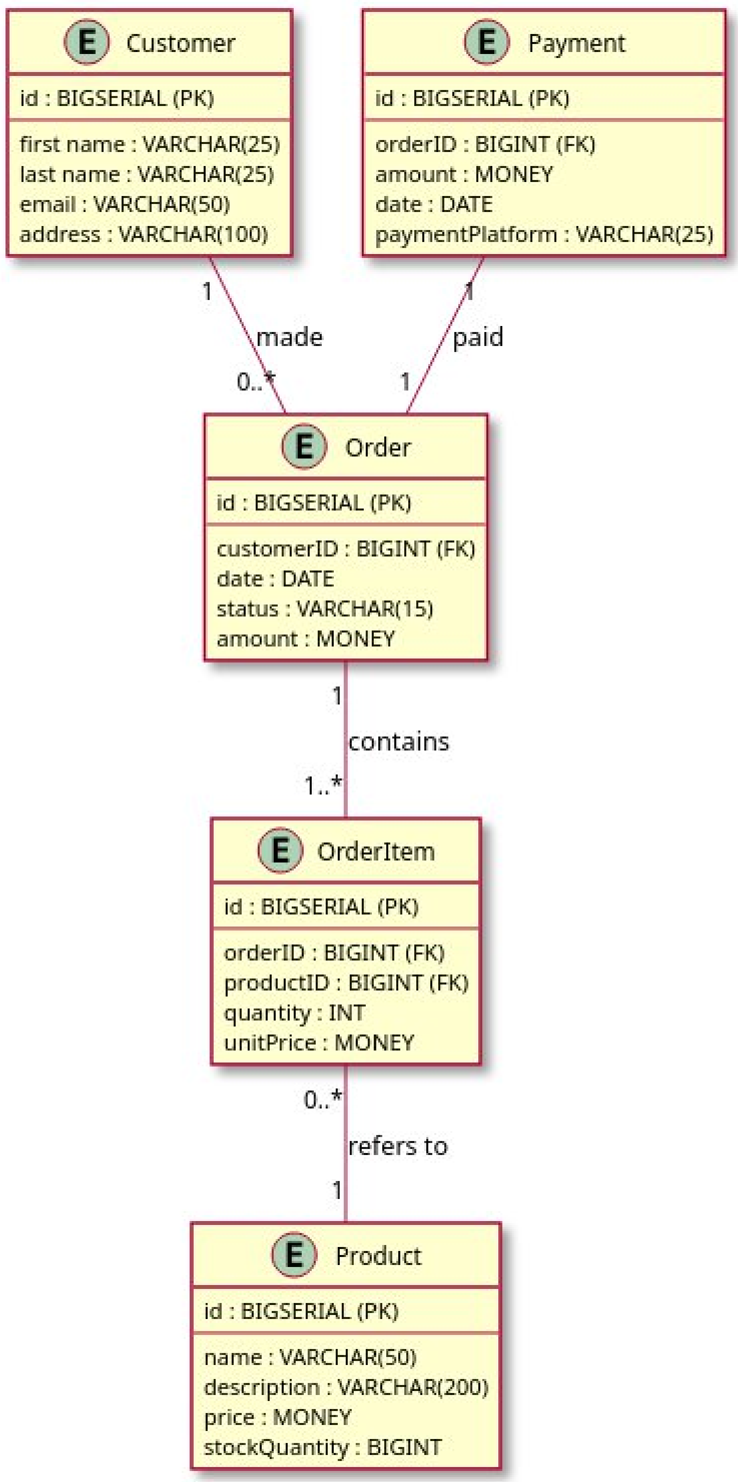
Ensure your diagram is clear, well-organized, and accurately represents the structure of the database.

Type of Diagram and Result

For this exercise, I used UML Class Diagram, using plantUML sytax code. Here is my script :

```
entity "Customer" {
    id : BIGSERIAL (PK)
    --
    first name : VARCHAR(25)
    last name : VARCHAR(25)
    email : VARCHAR(50)
    address : VARCHAR(100)
}
entity "Product" {
    id : BIGSERIAL (PK)
    --
    name : VARCHAR(50)
    description : VARCHAR(200)
    price : MONEY
    stockQuantity : BIGINT
}


entity "Order" {
    id : BIGSERIAL (PK)
    --
    customerID : BIGINT (FK)
    date : DATE
    status : VARCHAR(15)
    amount : MONEY
}


entity "OrderItem" {
    id : BIGSERIAL (PK)
    --
    orderID : BIGINT (FK)
    productID : BIGINT (FK)
    quantity : INT
    unitPrice : MONEY
}


entity "Payment" {
    id : BIGSERIAL (PK)
    --
    orderID : BIGINT (FK)
    amount : MONEY
    date : DATE
    paymentPlatform : VARCHAR(25)
}


Customer "1" -- "0..*" Order : made
Order "1" -- "1..*" OrderItem : contains
OrderItem "0..*" -- "1" Product : refers to
Payment "1" -- "1" Order : paid
```

This PlantUML script generates the following diagram :


E Customer
id : BIGSERIAL (PK)
first name : VARCHAR(25)
last name : VARCHAR(25)
email : VARCHAR(50)
address : VARCHAR(100)
E Payment
id : BIGSERIAL (PK)
orderID : BIGINT (FK)
amount : MONEY
date : DATE
paymentPlatform : VARCHAR(25)
1
made
0..*
1
paid
1
E Order
id : BIGSERIAL (PK)
customerID : BIGINT (FK)
date : DATE
status : VARCHAR(15)
amount : MONEY
1
contains
1..*
E OrderItem
id : BIGSERIAL (PK)
orderID : BIGINT (FK)
productID : BIGINT (FK)
quantity : INT
unitPrice : MONEY
0..*
refers to
1
E Product
id : BIGSERIAL (PK)
name : VARCHAR(50)
description : VARCHAR(200)
price : MONEY
stockQuantity : BIGINT

III – Documentation

Entities and attributes

Customer

id	BIGSERIAL PK	Unique identifier for each customer
firstName	VARCHAR(25)	Customer's first name
lastName	VARCHAR(25)	Customer's last name
email	VARCHAR(50)	Customer's email
address	VARCHAR(100)	Customer's address

Product

id	BIGSERIAL PK	Unique identifier for each product
name	VARCHAR(50)	Product name
description	VARCHAR(200)	Product description
price	MONEY	Price of the product
stockQuantity	BIGINT	Quantity available in stock for this product

Order

id	BIGSERIAL PK	Unique identifier for each order
customerId	BIGINT FK (customer table)	Id of the customer that made the order
date	DATE	Date of the order
status	VARCHAR(15)	Status of the order
amount	MONEY	Total amount of the order (sum of all products)

OrderItem

id	BIGSERIAL PK	Unique id for each item of an order
orderId	BIGINT FK (order table)	Id of the order containing the product
productId	BIGINT FK (product table)	Id of the product (eg : id of a corsair keyboard)
quantity	INT	Quantity of the product ordered
unitPrice	MONEY	Individual price of the product (by unit)

Payment

id	BIGSERIAL PK	Unique id of a payment
orderId	BIGINT FK (order table)	Id of the order associated with the payment
amount	MONEY	Total amount of the payment
date	DATE	Date of the payment
paymentPlatform	VARCHAR(25)	Platform used to pay (traditional card, PayPal, PaySafeCard...)

Relationships

Customer - Order :

- A customer can make 0 or many order
- An order is already made by one customer

Order – OrderItem :

- An order contains one or many items
- Each item belongs to one unique order

OrderItem – Product :

- An item is referred to only one product
- A product can appear to 0 or many items

Order - Payment :

- An order is associated with one unique payment
- A payment is associated with one unique order

Queries implemented

1 - All orders from a specific customer :

Retrieves every order that a customer made, using their email address (could also be with id)

```
SELECT
   c.firstName,
   c.lastName,
   o.id AS orderID,
   o.date AS orderDate,
   o.amount AS totalAmount
FROM
   customer AS c
JOIN
   "order" AS o ON c.id = o.customerID
WHERE
   c.email = 'jean.dupont@email.com';
```

2 – All items of a specific order

Retrieves all items included in an order using its order id.

```
SELECT
   oi.orderID,
   p.name AS productName,
   oi.quantity,
   oi.unitPrice
FROM
   orderitem AS oi
JOIN
   product AS p ON oi.productID = p.id
```

```
WHERE
   oi.orderID = 1;
```

3 – All products with low stocks

Retrieves every products with a stock less than 15

```
SELECT
   p.name AS productName,
   p.stockQuantity
FROM
   product AS p
WHERE
   p.stockQuantity < 15
ORDER BY
   p.stockQuantity ASC;
```

4 – Most trending products

Retrieve the orders that are the most popular to the customers (best sellers)

```
SELECT
   p.id,
   p.name,
   SUM(oi.quantity) AS totalOrdered
FROM
   product AS p
JOIN
   orderitem AS oi ON p.id = oi.productID
GROUP BY
   p.id, p.name
ORDER BY
   totalOrdered DESC
```

5 – Average order value

Average price of an order (according to the order made by all the clients)

```
SELECT
   ROUND(AVG(o.amount), 2) AS averageOrderValue
FROM
   "order" AS o;
```

Link to the GitHub Project

https://github.com/Stark3rn/croatia_sql_final_project