CST 2550: Software Engineering Management and Development

Group Coursework 2: Report

Autumn/Winter term 2024/2025

STUDENT NAMES	STUDENT ID NUMBER
AMEERA NAJLA MANJOO	M01014463
PAULINE JONAS MTALLO	M00937209
SHEJA DAVID	M00977005
MATHEWS MWANGI	M00836628

Lab Tutor: Miss Aisha Idoo

Date of submission : 16/04/25

Table of Contents

Table of Figures

DRIVING LESSON BOOKING SYSTEM

Introduction

Project Description

This project involved a driving lesson booking system built with C# and SQLite. The system aims to automate the fundamental processes in a driving school. It is composed of four entities, all of which are necessary for a successful booking entry. These entities are student, car, instructor, and booking entities. The functionalities facilitate the management of these entities where new instances can be added, an existing instance can be updated, deleted, or read. A linked list is used for effective sorting and a hash table is utilized to optimize the search processes. Besides these data structures, SQLite database is also implemented to persist the storage process thus providing a hybrid approach for information storage and retrieval.

Project Structure

The project begins with the **design process**, explaining the data structures and algorithms used and their analysis. In this section, the choice for the database structure is also highlighted through an entity relationship diagram (ERD). Following the design is the **testing section**. In this section, the testing approaches used are highlighted and the test case results are listed. A reflection of the development process is explained in-depth in the **conclusion** of the project and references used throughout the report are provided in the **references** section of the report.

Design

Justification

Hash Table

A hash table is very important in processes involving key-based lookups. In the case of the driving lessons booking system, it is used to implement all the entities with the key being the primary key of the entities. It is better because its time complexity is constant time (O (1)) on average with searches, insertions, and deletions (Libraries et al., 2021). This makes the BookManager, InstructorManager, StudentManager, and CarManager faster with searches, insertions, and deletions no matter how many records we might have in our database. To make the processes seamless, the information will be loaded from the database before the users' interactions.

Linked List

Linked list data structure is selected to arrays for the sorting processes. It is especially important when it comes to space complexity. Unlike arrays whose elements are placed contiguously in memory, linked lists have pointers thus offering flexibility in memory allocation as nodes can take any location in memory, not requiring contiguous blocks (Koyande, 2024). This in turn prevents wastages as memory doesn't require to be pre-allocated because the linked list grows or shrinks at run time. It thus works best for sorting the entities used in our system.

Analysis Using Pseudocode

Because all inserts follow the same complexity as our hash table, we will use the operation of one of the entities to analyze it.

Pseudocode

```
INSERT_INSTANCE (instance)  index ←
HASH (instance.ID) % TABLE_SIZE  IF
table[index] is EMPTY THEN  table[index]
← instance
  ELSE
    Traverse linked list at table[index]
    IF instance. ID EXISTS THEN
      UPDATE instance data
    ELSE
      Append instance at the end of the list
```

The above pseudocode shows the index computation that takes constant time (O(1)) in line 2. The first insertion operation also takes constant time if there is no existing student. In the case of collision, a linear time complexity will be assumed because of list traversal to look for the appropriate place to do the insertion. Search operation will also take a constant time on average but collisions may slow the process down to a linear time. With tables, lookups, insertion, and deletion are thus on average constant time but due to collisions, they may be linear (O(n)).

```
LOAD_INSTANCE_FROM_DB ()
```

Open SQLite connection

Execute "SELECT * FROM Instance"

FOR each record in result:

instance ← Create Instance Object

INSERT into Hash Table

INSERT into Sorted Linked List

The process of loading students from the database to accomplish our hybrid approach takes a linear time. This means that the time increases with the number of records. However, after loading the information into the hash table and linked list, the average constant time complexity should be easily achieved except in cases of collision.

Data Structures and Justification

Data Structure	**Usage**	**Justification**
Hash Table (CustomHashTable<string, Instance>)	Stores instance using InstanceID as key.	Provides **O (1) average time complexity** for lookups, inserts, and deletions.
Sorted Linked List (SortedLinkedList<Instance>)	Maintains an ordered list of instances.	Allows **efficient sorted retrieval** of instances for display.
SQLite Database	Persists data.	Saves data across sessions

Table 1: shows the different data structures used and how their algorithm operations compare.

Using SQL is important in persisting data. As much as the data structures are important, they can only allow memory usage. To effectively implement the database functionality, it is necessary to follow a good design to minimize redundancy and improve performance (Inzaugarat, 2025). Our design process entailed implementing an ERD to ensure the tables are correctly normalized as seen in Figure 1.

Database Design (ERD)

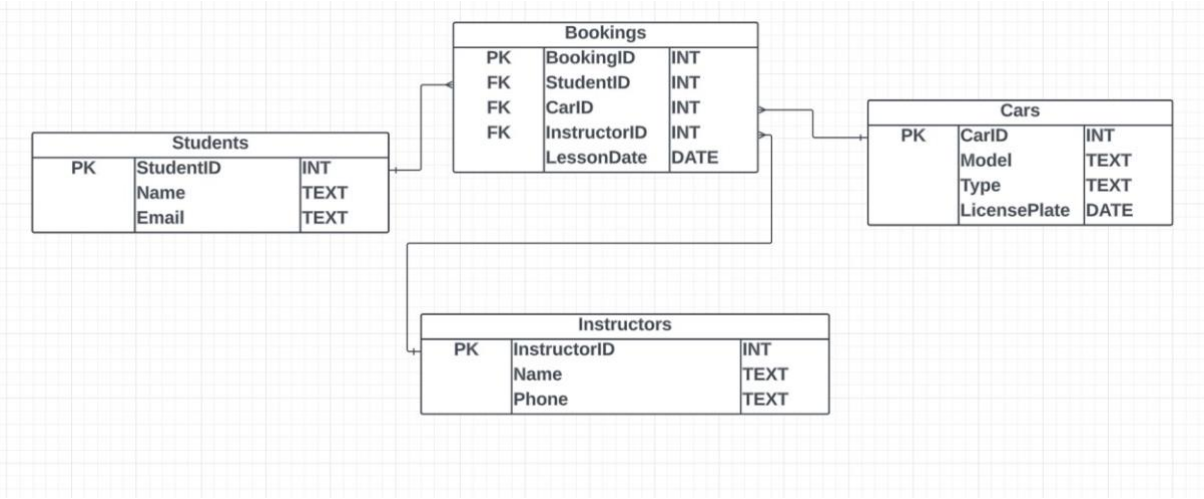

Figure 1: Shows the normalized tables for efficient information retrieval and improved performance.

Testing

We applied the MSTest suite in the testing process using three testing approaches to ensure a fully functional code. To ensure the functionality of individual functions, we did unit testing. Through these unit tests, we were able to get test outputs and ensure the functions were producing the desired results. For testing the data structures and algorithms, functional tests were performed. Through these tests, the different operations performed throughout the system were fully tested. For instance, when a student is added successfully, an output should be produced informing the user that their record was added successfully (Ncarandini et al., 2023). Edge cases were also tested to validate inputs, test missing files, and check for cases where duplicate IDs were provided for instances where there should be unique IDs. Table 2 shows the test cases table with inputs, expected outputs, and the results of the tests.

Test Cases Table

Test Case	**Input**	**Expected Output**	**Actual Output**	**Pass/Fail**
Add Student	StudentID = "S123", Name = "John"	Student added successfully	Student added successfully	Pass
Add Duplicate Student	StudentID = "S123" again	Error: Student already exists	Error: Student already exists	Pass

Delete Student	StudentID = "S123"	Student deleted	Student deleted	Pass
Search Non-Existent Student	StudentID = "S999"	Error: Student not found	Error: Student not found	Pass

Table 2: Different test cases and their results

Conclusion

In **summary**, we implemented different data structures and algorithms to aid in performing different operations for our driving lessons booking system. For ordered retrieval, we used a sorted linked list while using a hash table for the other operations. To fully improve the performance of the operations while persisting the data across sessions, we implemented an SQLite database and performed different testing strategies to ensure the expected output.

Some of the limitations we encountered during the project were collisions that were caused due to poor hash functions. In some cases, we encountered errors due to negative calculated values leading us to use the absolute function to ensure only positive hashes. Using SQLite caused locking issues due to simultaneous database accesses. Table 3 shows a summary of these limitations.

Limitations and Reflections

Limitation	**Cause**
Collisions	Due to poor hash function in some cases with the hash table (Joshi, 2017).
Locking issues	Happened with SQLite queries accessing DB simultaneously (Rathbone, 2024).

Table 3: Limitations and reflections

For future improvements, we would implement the database using an SQL server because of its high performance which in turn helps optimize query execution. We will also incorporate better hash functions to prevent data collisions when using a hash map and a linked list as the data structures to manage our system application. Database performance is also key. Indexing optimizes performance and will be our key consideration in future projects.

References

Inzaugarat, M.E. (2025) *SQL query optimization: 15 techniques for better performance*. DataCamp. Available at: https://www.datacamp.com/blog/sql-query-optimization (Accessed: 14 March 2025).

Joshi, V. (2017) *Taking hash tables off the shelf*. Medium. Available at: https://medium.com/basecs/taking-hash-tables-off-the-shelf-139cbf4752f0 (Accessed: 12 March 2025).

Koyande, V. (2024) *Arrays vs. linked lists*. Medium. Available at: https://medium.com/@vedantikoyande/arrays-vs-linked-lists-726b70410132 (Accessed: 10 March 2025).

Libraries, L. (2021) *7.1: Time complexity and common uses of hash tables*. Engineering LibreTexts. Edited by N.Cx. Expert, U. Davis, and MERLOT. Available at: https://eng.libretexts.org/Courses/Folsom_Lake_College/CISP_430%3A_Data_Structures_(Aljuboori)/07%3A_Hash_Tables/7.01%3A_Time_complexity_and_common_uses_of_hash_tables#:~:text=Hash%20tables%20are%20often%20used,schemes%20is%20O(n). (Accessed: 14 March 2025).

Ncarandini et al. (2023) *Unit testing C# with MSTest and .NET*. Microsoft Docs. Available at: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest (Accessed: 11 March 2025).

Rathbone, M. (2024) *Understanding and resolving the "SQLite database is locked" error*. Beekeeper Studio Blog. Available at: https://www.beekeeperstudio.io/blog/how-to-solve-sqlite-database-is-locked-error#:~:text=The%20%E2%80%9CSQLite%20database%20is%20locked%E2%80%9D%20error%20is%20typically%20a%20symptom,of%20database%20connections%20and%20transactions. (Accessed: 14 March 2025).