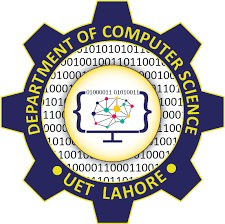
**ADVANCE DATABASE MANAGEMENT SYSTEM**

****

**Session 2023 – 2027**

**Submitted By:**

Asad Tayyab 2023-CS-14

Sheharyar Zahid 2023-CS-30

Munees Tariq 2023-CS-32

Syed M. Zaeem 2023-CS-38

**Supervised By:**

Dr. Aatif Hussain

**Course:**

CSC-207 Advance Database Management System

**Department Of Computer Science**

**University Of Engineering And Technology**

**Lahore, Pakistan**

Contents

**[1. Academic Submission](#_Toc198068899)** [4](#_Toc198068899)

**[2. Introduction](#_Toc198068900)** [4](#_Toc198068900)

**[3. Project Overview](#_Toc198068901)** [5](#_Toc198068901)

**[4. Objectives of the System](#_Toc198068902)** [6](#_Toc198068902)

**[5. Key Features](#_Toc198068903)** [6](#_Toc198068903)

**[6. Technologies Used](#_Toc198068904)** [7](#_Toc198068904)

**[7. System Design](#_Toc198068905)** [7](#_Toc198068905)

**[8. Database Structure](#_Toc198068906)** [8](#_Toc198068906)

**[8.1 File Storage Types](#_Toc198068907)** [8](#_Toc198068907)

**[9. User Interface Design](#_Toc198068908)** [9](#_Toc198068908)

**[9.1 Main Features of the UI](#_Toc198068909)** [9](#_Toc198068909)

**[10. Overview of Functionality](#_Toc198068910)** [10](#_Toc198068910)

**[11. How the Application Works](#_Toc198068911)** [10](#_Toc198068911)

**[11.1. Start Form / Home Screen](#_Toc198068912)** [10](#_Toc198068912)

**[11.2. Forms & Controls](#_Toc198068913)** [11](#_Toc198068913)

**[12. Backend Logic & Code Flow](#_Toc198068914)** [11](#_Toc198068914)

**[12.1 Adding a Record](#_Toc198068915)** [11](#_Toc198068915)

**[12.2](#_Toc198068916)****[Searching a Record](#_Toc198068916)** [12](#_Toc198068916)

**[12.3 Deleting a Record](#_Toc198068917)** [12](#_Toc198068917)

**[12.4 Updating a Record](#_Toc198068918)** [12](#_Toc198068918)

**[12.5 Displaying All Records](#_Toc198068919)** [13](#_Toc198068919)

**[13. Error Handling](#_Toc198068920)** [13](#_Toc198068920)

**[14. Class Structure](#_Toc198068921)** [13](#_Toc198068921)

**[15. Tables and Records (Just Like SQL Tables)](#_Toc198068922)** [14](#_Toc198068922)

**[16. Query Execution](#_Toc198068923)** [14](#_Toc198068923)

**[16.1 Select Queries](#_Toc198068924)** [14](#_Toc198068924)

**[16.2 Where Conditions](#_Toc198068925)** [14](#_Toc198068925)

**[16.3 Update Queries](#_Toc198068926)** [15](#_Toc198068926)

**[16.4 Delete Queries](#_Toc198068927)** [15](#_Toc198068927)

**[17. Joins (Connecting Tables)](#_Toc198068928)** [15](#_Toc198068928)

**[18. Transaction Manager](#_Toc198068929)** [16](#_Toc198068929)

**[18.2 What Is It?](#_Toc198068930)** [16](#_Toc198068930)

**[18.3 How It Works in the Project:](#_Toc198068931)** [16](#_Toc198068931)

**[18.4 Why It’s Important?](#_Toc198068932)** [16](#_Toc198068932)

**[19. Summary of Advanced SQL Features Implemented](#_Toc198068933)** [17](#_Toc198068933)

**[20. Main Folders and What They Contain](#_Toc198068934)** [17](#_Toc198068934)

**[20.1 MainCode](#_Toc198068935)** [17](#_Toc198068935)

**[20.2 bin and obj](#_Toc198068936)** [17](#_Toc198068936)

**[20.3 Data](#_Toc198068937)** [18](#_Toc198068937)

**[21. How to Navigate the System (Step-by-Step)](#_Toc198068938)** [18](#_Toc198068938)

**[21.1. Launch the Application](#_Toc198068939)** [18](#_Toc198068939)

**[21.2. Home Screen](#_Toc198068940)** [18](#_Toc198068940)

**[21.3. Perform Actions](#_Toc198068941)** [18](#_Toc198068941)

**[22. User-Friendly Features](#_Toc198068942)** [19](#_Toc198068942)

**[23. Developer Notes](#_Toc198068943)** [19](#_Toc198068943)

**[24. Example Navigation Flow](#_Toc198068944)** [20](#_Toc198068944)

**[25. Access Control and Validations](#_Toc198068945)** [20](#_Toc198068945)

**[26. Indexing](#_Toc198068946)** [20](#_Toc198068946)

**[27. Query Parser](#_Toc198068947)** [21](#_Toc198068947)

**[28. Future Plans for the System](#_Toc198068948)** [21](#_Toc198068948)

**[28.1 Planned Improvements](#_Toc198068949)** [21](#_Toc198068949)

**[29. What We Learned](#_Toc198068950)** [22](#_Toc198068950)

**[29.1](#_Toc198068951)****[Technical Skills](#_Toc198068951)** [22](#_Toc198068951)

**[29.2 Logical Thinking](#_Toc198068952)** [22](#_Toc198068952)

**[29.3 Teamwork & Time Management](#_Toc198068953)** [22](#_Toc198068953)

**[30. Known Limitations / Flaws](#_Toc198068954)** [22](#_Toc198068954)

**[31. Final Conclusion](#_Toc198068955)** [23](#_Toc198068955)

**[Wireframes](#_Toc198068956)** [24](#_Toc198068956)

# **1. Academic Submission**

This document presents a complete overview of our project titled **"Advanced Database Management System (ADBMS)"**. It is a full-fledged database solution developed using the C# programming language. The project has been designed and implemented with real-world requirements in mind, not as a student assignment or learning prototype, but as a robust application capable of managing different types of data across various fields.

The system combines multiple use cases into a single integrated software, offering independent but coordinated modules for different types of databases. These include systems for airline bookings, inventory management, school administration, course tracking, and more. The project demonstrates a real-time implementation of many key database concepts such as data storage, retrieval, editing, searching, querying.

This submission outlines the full architecture, functionality, and working of the system using simple language that’s easy to follow, while also maintaining technical accuracy for academic and professional presentation.

# **2. Introduction**

In today's world, data is one of the most valuable assets for any organization. Whether it's a business, school, or service provider, proper data management ensures smooth operations, good decision-making, and reliable services. The main goal of our project was to create a powerful and flexible database system that can manage a variety of data types under one application.

This Advanced Database Management System (ADBMS) is not limited to a single domain. It supports multiple sectors like airline services, inventory management, school systems, and more. The project is created using C# with Windows Forms for the user interface, and file handling techniques such as CSV and binary file operations for data storage.

The system provides individual modules that are easy to access and operate. Each part functions independently while maintaining a similar layout and experience across the whole application. This makes it easier for users to learn and use different parts of the system with minimal training.

# **3. Project Overview**

This project is a complete and professional-level **Advanced Database Management System (ADBMS)** built using **C#** and **Windows Forms**. It is created to manage and store different types of data in a clean, organized, and user-friendly way. Unlike student projects or demos, this system is designed to be scalable, modular, and flexible enough to handle real-world requirements in various domains.

The main goal of the ADBMS project is to demonstrate core database concepts in action. It allows users to perform typical database operations (like creating tables, adding or deleting records, and querying data) through a graphical interface. It also implements advanced features such as transactions and indexing to make data operations reliable and efficient. In summary, this project is a Windows desktop application (using Microsoft Windows Forms) that simulates an all-in-one database management system, aimed at illustrating how databases can be managed and queried.

# **4. Objectives of the System**

The purpose behind this project was to build a database system that goes beyond the basic CRUD operations. We wanted to create something that could actually help in real-world situations. The first objective was to provide a unified platform where different sectors can manage their own data without needing separate tools for each one.

Another main goal was to make the software user-friendly. We used Windows Forms in C# to design simple and clean interfaces so even non-technical users can work with the system. We also focused on modularity. This means each part of the software can be developed, tested, or replaced without affecting the rest of the system.

Data validation, storage, and retrieval were also key objectives. The system ensures that wrong data doesn't get stored and that every record can be easily accessed when needed. This is handled through in-built checks and error-handling code in each form. Finally, we aimed to give users a smooth experience, with proper navigation, form resets, confirmations, and alerts so that they always know what’s happening in the system.

# **5. Key Features**

* Includes **practical features** for efficient data management
* Uses both **binary and CSV files** for fast and lightweight data storage
* Each form supports: **Add**, **update**, **delete**, and **search** operations
* Features **clear navigation** with menus and labeled buttons
* Designed to ensure **simple, fast, and error-free** data handling

# 

# **6. Technologies Used**

* Built using **C#** as the core language
* **Windows Forms** used for user interface design
* **Binary files** store complex data efficiently
* **CSV files** allow easy data viewing in tools like Excel
* **Local file storage** enables offline use without a database server

# **7. System Design**

The design of this project is based on the concept of **modular architecture**. This means the system is divided into different parts (called modules), and each part works on its own specific data but follows the same structure. This approach makes it easier to manage, update, or even replace one part of the system without breaking the rest.

Each module in the system contains:

* A user interface (form) where the user can enter or view data
* A data file (CSV or binary) to store the records
* Logic to handle adding, editing, deleting, and searching data

The system follows a **simple flow**:

1. User opens the main application window.
2. From the menu, the user selects the desired database or create a new database.
3. The selected database opens.
4. The user interacts with buttons, text fields, and data tables to manage records.
5. Data is saved or loaded from the file whenever required.

This clean structure allows us to expand the system in the future by just adding more modules, without changing the overall layout or code.

# **8. Database Structure**

Instead of using a large external database like MySQL or SQL Server, this system mostly uses **file-based storage**. Data is stored locally either in **CSV files** (comma-separated values) or **binary files**. Each module has its own file where it keeps its records.

## **8.1 File Storage Types**

* **CSV Files**  
  These are text files where each row is a record, and values are separated by commas. They are easy to open in Excel and are great for reading/writing simple data.
* **Binary Files**  
  These store data in a compact, unreadable format that is faster to load and saves space. Binary files are useful when we want to prevent users from directly editing the data.

# **9. User Interface Design**

The system is built using **Windows Forms**, which is a feature of C# that allows us to design desktop applications with buttons, text boxes, labels, and menus.

## **9.1 Main Features of the UI**

* **Easy Navigation**  
  The main window provides a clear menu to select the desired module. Each form is labeled properly, and buttons have clear text to show what they do (like "Add", "Delete", "Search", etc.).
* **Consistent Design**  
  All forms follow the same layout pattern. This makes it easier for users to understand and use different parts of the system without confusion.
* **Data Entry Forms**  
  Each module has forms with input fields where the user can enter data. These forms include:
  + Text boxes for typing names, IDs, etc.
  + Drop-downs for choices.
  + Buttons to perform actions like saving or updating data
* **Validation**  
  The system checks that the user fills all the required fields before saving the data. If anything is missing or incorrect, an error message is shown.
* **Feedback Messages**  
  After each action (like saving or deleting), a message appears to tell the user whether it was successful or not.
* **Search and Display**  
  Many modules allow users to search for data. The results are shown in a list or table view, making it easy to find and review records.

This easy interface helps users perform data operations without needing technical knowledge.

# **10. Overview of Functionality**

The Advanced Database Management System (ADBMS) is a full application developed using **C# Windows Forms**. The main purpose is to allow users to perform database-related operations such as:

* Database CRUD using UI.
* Query Parser.
* Transactions Manager.
* Indexing for faster access.
* Easy to use UI/UX.

# **11. How the Application Works**

## **11.1. Start Form / Home Screen**

* When the application starts, the **Main Form** appears.
* It shows the options for the user to choose what part of the system they want to work with.
* After selecting, a new window opens, where the user can add, view, or manage the data.

This design keeps the user interface clean and separates each task properly.

## **11.2. Forms & Controls**

Every task is handled through a form. The forms include:

* TextBoxes (for input fields)
* Buttons (Add, Update, Delete, Search)
* DataGridView (to display data in table form)
* MessageBoxes (to show feedback like “Record Added Successfully”)

These forms are **connected to backend logic** which handles what happens when a button is clicked.

# **12. Backend Logic & Code Flow**

Now let’s understand what happens behind the scenes when the user performs different actions.

## **12.1 Adding a Record**

When the user enters data and clicks the **"Add"** button:

* The system checks if all required fields are filled.
* This object is then converted into a string (for CSV) or written directly as binary.
* The record is saved into a file using FileStream, StreamWriter, or BinaryWriter.

Success Message is shown using MessageBox.Show("Record Added").

## **12.2** **Searching a Record**

When the user wants to search for a specific entry:

* They enter the search value (like Name, ID, etc.) in a textbox.
* On clicking **"Search"**, the system opens the file and reads each record line by line.
* It checks if the line contains the matching value.
* If found, the data is displayed on screen using DataGridView.

If not found, a message is shown:  
"No matching record found."

## **12.3 Deleting a Record**

When the user chooses to delete a record:

* The system reads the file and creates a **temporary copy** that skips the matching record.
* The original file is deleted.
* The new file is renamed to take its place.

This method makes sure the system only removes what’s needed

## **12.4 Updating a Record**

To update any record:

* The user first searches for the record.
* After it's loaded into the form, the user makes the necessary changes.
* The system again creates a temporary file with updated values.
* The updated file replaces the old one.

This approach prevents direct file edits, reducing the risk of errors.

## **12.5 Displaying All Records**

To show all saved data:

* The system opens the file and reads all records line by line.
* Each record is split and shown in a table (using DataGridView).
* The table makes it easy to view large sets of data quickly.

This is helpful for reviewing or printing records.

# **13. Error Handling**

The application also includes **basic error checking**, such as:

* Making sure no fields are left empty
* Preventing duplicate entries
* Showing proper messages when a file is missing or corrupted

This helps make the application more **stable and user-friendly**.

# 

# **14. Tables and Records (Just Like SQL Tables)**

In this system, each set of data is treated like a **table** — similar to how we see tables in SQL databases.

* Each "table" is represented by a **file** (CSV or binary).
* Each line in the file is one **record** (like a row in SQL).
* Each field (like ID, Name, Age) is like a **column**.

these files can be:

* Opened
* Searched
* Updated
* Deleted  
  ...just like SQL tables using SELECT, UPDATE, or DELETE.

# **16. Query Execution**

Even though this system does not use a real SQL engine, it mimics SQL behavior very closely using **custom code logic**.

## **16.1 Select Queries**

The user can view all records from a table, which works like:

SELECT \* FROM Employees;

## **16.2 Where Conditions**

The system allows filtering based on specific values:

SELECT \* FROM Employees WHERE Department = 'HR';

This is done by searching each record in the file and checking conditions using if statements in C#.

## **16.3 Update Queries**

When updating a record, the logic is similar to:

UPDATE Employees SET Salary = 60000 WHERE ID = 101;

Your C# code reads each line, finds the matching ID, and writes back the new data.

## **16.4 Delete Queries**

When deleting, it's like:

DELETE FROM Employees WHERE Name = 'Ali';

Your system skips the line that matches this condition and rewrites the rest into the new file.

# **17. Joins (Connecting Tables)**

Just like SQL allows combining data from two tables, your project supports similar functionality using **joins**.

Example: If you have an "Employees" file and a "Departments" file:

* The system can match the **Department ID** from Employees with **ID** in Departments.
* It shows combined results — for example, employee names along with their department names.

In C#, this is achieved by:

* Reading both files
* Matching related fields using nested loops
* Displaying combined data in a table view

This gives users the power of **relational database logic** without needing a database server.

# **18. Transaction Manager**

The project also includes a **Transaction Manager** — a very advanced and professional feature.

## **18.2 What Is It?**

A transaction manager controls a set of operations that must either:

* All happen together, or
* Not happen at all (rollback on error)

This prevents situations like partial updates or data corruption.

### **18.3 How It Works in the Project:**

1. **Begin Transaction**:  
   The system starts recording all operations (like add, update).
2. **Commit**:  
   If all operations are successful, the system finalizes all changes.
3. **Rollback**:  
   If any error occurs, all operations are cancelled, and data goes back to its original form.

## **18.4 Why It’s Important?**

* It helps maintain **data integrity**
* Makes the system more **reliable**
* Allows multiple changes without worrying about crashes or mistakes

Our C# code uses temporary files and in-memory objects to manage this rollback safely.

# **19. Summary of Advanced SQL Features Implemented**

| **Feature** | **Equivalent in Our Project** |
| --- | --- |
| Tables | File-based records |
| SELECT | Read and display records |
| WHERE | Filter using conditions |
| UPDATE | Modify records in file |
| DELETE | Remove record from file |
| JOIN | Match and merge two tables |
| TRANSACTIONS | Group changes safely |
| ROLLBACK / COMMIT | Cancel or confirm changes |

# **20. Main Folders and What They Contain**

## **20.1 MainCode**

This is the heart of the project. It holds the actual code files that run the application. It includes:

* Form files (.cs): User interface logic
* Classes (.cs): For handling operations like file reading, transactions, etc.
* Logic files: Handle add/update/delete/query actions

## **20.2 bin and obj**

These folders are automatically generated by Visual Studio and are used for **compiling and running** the project. You don’t need to touch these unless debugging.

## **20.3 Data**

This folder contains the **data files** used as tables. These can be:

* .csv files: for readable tables
* .bin files: for faster, binary data handling

Each file acts like a SQL table, storing multiple rows and columns of information.

# **21. How to Navigate the System (Step-by-Step)**

## **21.1. Launch the Application**

* Run the project from Visual Studio or use the compiled .exe file in bin/Debug.
* The system opens in a user-friendly **graphical interface** (forms-based).

## **21.2. Home Screen**

* The home screen shows buttons or menus like:
  + **Create New Table**
  + **Run Query**
  + **Transaction Manager**
  + **Exit**

## **21.3. Perform Actions**

Each action you click opens a new form. For example:

#### **Add Record**

* Choose a table
* Fill in details
* Click “Add”

#### **Update or Delete**

* Search for a record using filters
* Click “Edit” or “Delete” to perform the action

#### **Run Query**

* Use simple SQL-like statements (example: SELECT \* FROM Employees)
* View results in a data grid

# **22. User-Friendly Features**

* All actions are performed using **buttons and forms**
* Input fields guide the user on what to type
* Error messages appear if anything is missing or wrong
* Queries give clear results in tables
* No need for users to know SQL — just select and click

# **23. Developer Notes**

If another developer wants to understand or modify your system, here are tips:

* All **form code** is in .cs files (like Form1.cs)
* Each class is written with clear function names (like InsertData(), DeleteByID())
* The system uses **OOP principles** — each task is split into methods and classes
* File I/O operations are clean and reusable
* Common utilities like searching or filtering are placed in **reusable helper functions**

# **24. Example Navigation Flow**

Here’s how a typical session might look:

1. Open app → Home screen
2. Click “Create Table” → Enter table name and fields
3. Click “Insert Record” → Add multiple records
4. Click “Run Query” → Type query like SELECT \* FROM table WHERE ID = 1
5. View results → Export or save if needed
6. Click “Exit” → Done

# **25. Access Control and Validations**

* System checks if file (table) exists before operating
* Prevents duplicate records using ID or key fields
* Handles wrong inputs with error messages
* Keeps files safe from corruption using try-catch and backups

# **26. Indexing**

The system also includes **indexing** to make data access faster and more efficient. When a user searches for records (especially using IDs or keys), the system uses indexing to quickly locate the required row instead of checking every record one by one. This makes search operations faster, especially in large tables. Indexes are managed internally while reading and writing files, helping improve overall performance during query execution.

# **27. Query Parser**

The system includes a **built-in query parser** that allows users to run SQL-like commands directly inside the application. This parser reads and understands basic query formats such as SELECT, INSERT, UPDATE, and DELETE. For example, users can write commands like SELECT \* FROM table WHERE id = 5, and the system will break down the statement, identify the table name, the action, and the condition, then perform the correct operation. It mimics how real database systems process queries, giving users a familiar way to interact with data without needing to click through multiple forms.

# **28. Future Plans for the System**

Even though the current system is functional and handles a lot of real-world database tasks, we have many ideas for future improvements. These will help make the system more powerful and professional.

## **28.1 Planned Improvements**

* **Graphical Reports:** Show charts and graphs based on table data.
* **SQL Editor:** Add a mini editor with syntax highlighting for writing queries.
* **Better Error Handling:** More detailed and helpful error messages for users.
* **Multi-user Support:** Enable multiple users to access the database at the same time.
* **Export to SQL:** Let users export tables into SQL format for other systems.

# **29. What We Learned**

While working on this project, we gained real-world experience in several important areas of software and database development:

## **29.1** **Technical Skills**

* Writing complete systems in **C# Windows Forms**
* Creating, updating, and reading **CSV and binary files**
* Building **custom SQL-like query support**
* Understanding how **joins** and **transactions** work
* Handling **file operations**, **error checking**, and **input validation**

## **29.2 Logical Thinking**

* How to break complex problems into small reusable functions
* Designing readable and scalable code
* Thinking like a real database system designer

## **29.3 Teamwork & Time Management**

* Dividing tasks among members
* Testing and merging features from different members
* Managing deadlines and bugs under pressure

# **30. Known Limitations / Flaws**

Every project has room for improvement. Here are some of the things we noticed:

* No support for large datasets (file handling can be slow with thousands of records)
* No database security (no encryption, no user roles)
* Manual field entry (no drag-and-drop form designer for tables)
* Query engine is basic — doesn’t support complex nested queries or functions
* UI can feel old-fashioned without modern styling

We aim to fix or improve these areas in the next versions.

# **31. Final Conclusion**

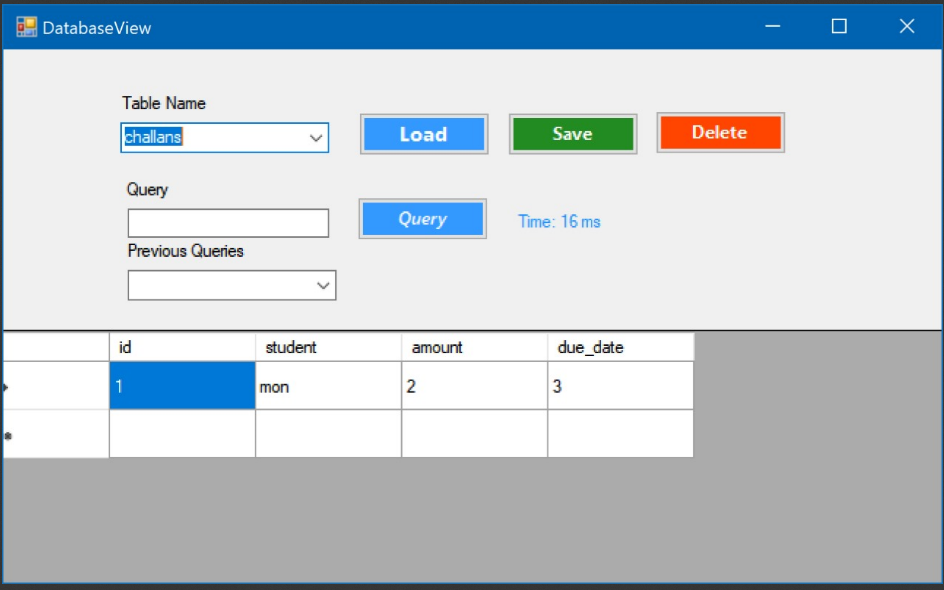
This Advanced Database Management System was a great step forward in building a working, real-world application from scratch. It is **not just a learning project** — it’s a complete system that shows how database engines can work using file handling, queries, joins, and transaction logic.

While it's not using SQL Server or MySQL behind the scenes, it still provides most features that any user expects from a small DBMS.

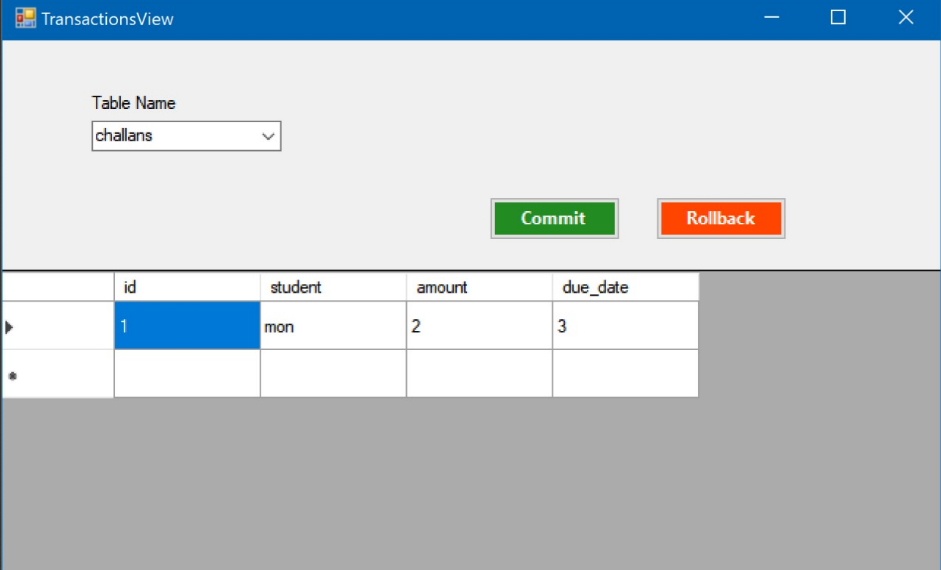
We are proud of what we built, and we know it can grow into something even more powerful with time.

“From code to query, from form to function – this project turned our ideas into reality.”

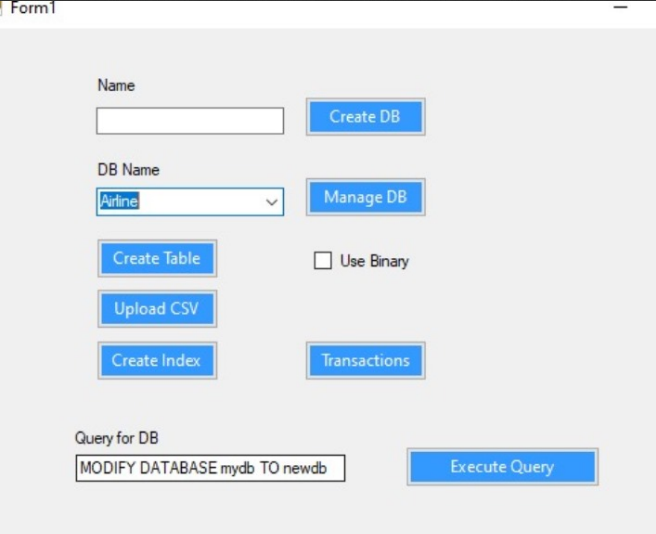
# **Wireframes**



DB View



Transection Management



Main View

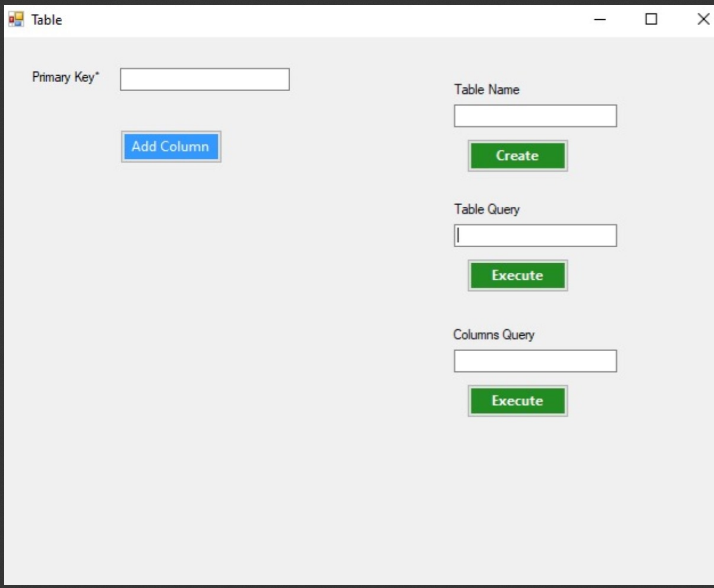


Table Manipulation