BlazeDB Custom Database Management System

Session 2023 - 2027

Submitted by:

Salman Anas	2023-CS-06
Amir Hashmi	2023-CS-11
Muhammad Talha	2023-CS-12
Mahad Sheraz	2023-CS-36

Supervised by:
Dr. Atif Hussain

Course:
CSC-207 Advanced DBMS

Department of Computer Science

University of Engineering and Technology
Lahore Pakistan

Table of Contents

1. Introduction

1.1 Project Overview

BlazeDB is a lightweight, high-performance custom Database Management System (DBMS) built entirely from scratch. Designed with educational and experimental objectives in mind, it demonstrates how core components of a DBMS such as data storage, query processsing, indexing, and front-end interaction work together. It features a custom file-based storage engine, B+ Tree indexing for efficient data retrieval, and a simplified SQL-like query language that supports essential CRUD operations and joins.

A lightweight, high-performance DBMS built from scratch featuring:

- Custom file-based storage engine
- B+ Tree indexing (O(log n) operations)
- SQL-like query language with JOINs
- React-based GUI with query editor

1.2 Tech Stack

- Backend: C++ – Provides high performance, direct memory management, and fine-grained control over I/O.

- Frontend: React.js and Node.js – Enables a responsive, user-friendly web interface for interacting with the database.

- Indexing: B+ Trees – Ensures efficient O(log n) performance for data retrieval and manipulation.

1.3 Key Features

Feature	**Description**
Storage Engine	Tables stored in .dat (data) and .idx (index) files
Query Language	Supports CREATE, INSERT, SELECT, UPDATE, DELETE, JOIN
Indexing	B+ Tree for fast lookups and CRUD operations in O (log n)

CRUD Operations	CREATE, DROP (databases & tables)
	INSERT, UPDATE, DELETE, SELECT (with WHERE clauses)

Joins	Joins for multi-table queries

Feature	Description
Logical Operators	Supports AND, OR, NOT, LIKE operators
Relational Integrity & Data Types	Primaryand Foreign Key enforcement with multiple data types (INT, VARCHAR, FLOAT, BOOL)
Frontend	Interactive query editor with syntax highlighting

2. System Architecture

2.1 Backend Components

The backend of BlazeDB is built using C++ and is designed with modularity in mind. Each component is responsible for a specific part of the database’s inner workings. These components work together to handle user queries, manage data, and maintain performance and reliability. Here's a detailed look at each one:

- **Database Manager**

  The Database Manager is the core controller of the system. Think of it as the brain that coordinates all high-level operations. Whether it's creating a new table, inserting data, or deleting a database, the Database Manager is responsible for ensuring the correct execution of these operations. It maintains a catalog of all tables and their schemas, enforces constraints (like primary and foreign keys), and updates internal metadata as the database evolves.

  In simple terms: If the database were a factory, the Database Manager would be the factory manager – assigning tasks, checking rules, and making sure everything runs smoothly.

- **Query Parser**

  The Query Parser is like the language interpreter of the system. When a user writes a SQL-like command (e.g., SELECT * FROM users;), it’s the parser's job to break that command down into understandable parts. It performs lexical analysis (breaking the query into tokens), syntactic analysis (checking grammar), and then translates the query into actions the database can understand.

In short: The Query Parser turns user commands into executable instructions for the backend – it bridges human language and system logic.

- **Storage Manager**

  The Storage Manager deals directly with the files on disk. It decides how data is physically stored, where each record is written, and how to read and write it efficiently. It handles file formats like .dat for table data, .idx for indexes, and .bin for metadata. It ensures that data is organized in fixed-size blocks (e.g., 4KB), and that all read/write operations are reliable and fast.

- **Index Manager**

  The Index Manager is responsible for maintaining B+ Tree indexes for primary keys. These indexes make data access faster, especially during search, insert, and delete operations. Instead of scanning the entire table, the Index Manager helps the system jump directly to the right spot using a sorted structure that enables quick lookups.

2.3 Frontend Components

The frontend of BlazeDB is built using modern web technologies – primarily React.js and Node.js – to provide a responsive and user-friendly interface for interacting with the database. It simplifies database operations for users by hiding complex backend processes behind a clean and intuitive UI. Below are the key components of the frontend:

- **Query Editor**

  The Query Editor is the main interface where users can write and execute their SQL-like queries. It features syntax highlighting to help users easily spot errors, keywords, and parameters. The editor is designed to be interactive and beginner-friendly, allowing users to perform database operations like creating tables, inserting data, and running JOIN queries – all through a web browser.

  In simple terms: It acts like a smart notepad that understands SQL and lets you talk to your database in real time.

- Result Viewer

  The Result Viewer displays the output of executed queries in a paginated table format. Whether you're retrieving ten rows or a thousand, it ensures that the results are cleanly displayed and easy to scroll through. Pagination makes it efficient to navigate large datasets without overwhelming the user.

3. Transaction Manager

3.1 Introduction to Transactions

A transaction is a logical unit of work that consists of one or more operations on the database. Transactions are crucial in multi-user database systems to ensure data correctness and consistency in the presence of concurrent access, failures, or unexpected interruptions.

3.2 Role in DBMS

The Transaction Manager in BlazeDB is responsible for tracking the beginning, processing, and termination (commit or rollback) of transactions. It ensures that the system can recover from crashes and handle simultaneous transactions without conflict. It works closely with the Storage Manager and Index Manager to guarantee that no partial or corrupt operations are left in the database.

3.3 ACID Properties

The core responsibility of the Transaction Manager is to enforce the ACID properties

Atomicity – All operations of a transaction are executed as a single unit. If one fails, the whole transaction is aborted, and the database is restored to its original state.

Consistency – Every transaction brings the database from one valid state to another. All integrity constraints must be preserved after every transaction.

Isolation – Concurrent transactions must not interfere with each other. BlazeDB ensures that the execution of one transaction is isolated from others, mimicking serial execution.

Durability – Once a transaction is committed, its changes are permanent, even in the case of system failure. This is ensured through logs and backups that the Transaction Manager maintains.

4. Storage Engine

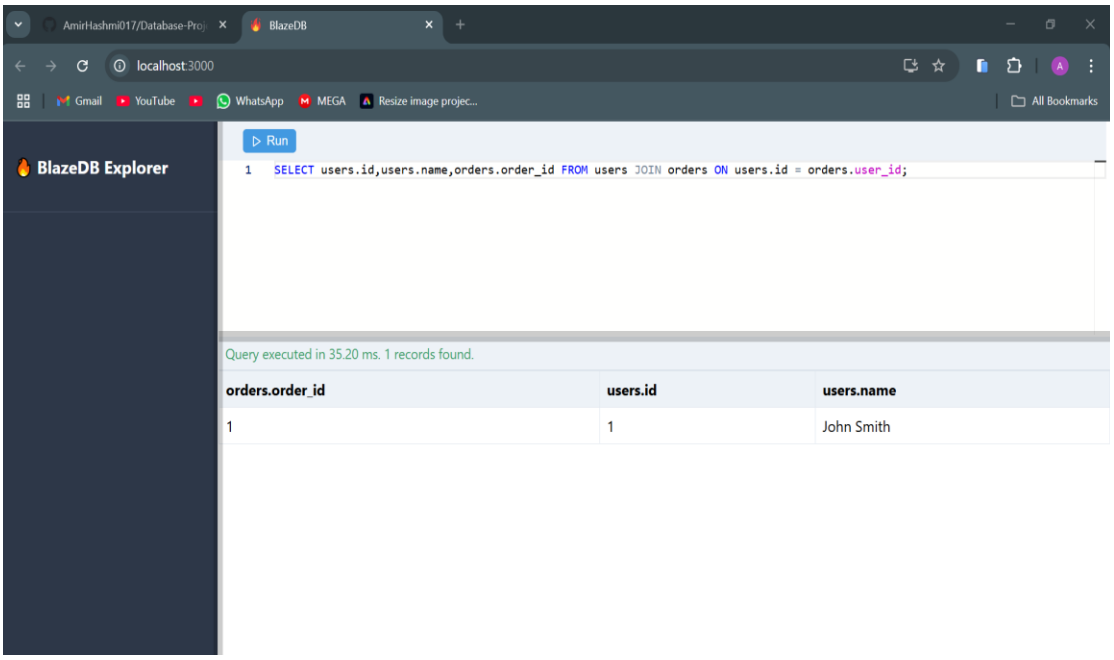
4.1 File Structure

```
mydb/   (Folder of Database)
├── catalog.bin      (Stores Database MetaData i.e Table's Schema)
├── employees.dat    (Stores Table's Records)
└── employees.idx    (Stores B+ Tree Indexes of Table's Primary Key)
```

4.2 Data Formats

File Type	Structure	Purpose
.dat	Fixed-size blocks (4KB)	Row storage
.idx	B+ Tree nodes (serialized)	Indexing
.bin	Metadata	Table metadata

5. Wireframes

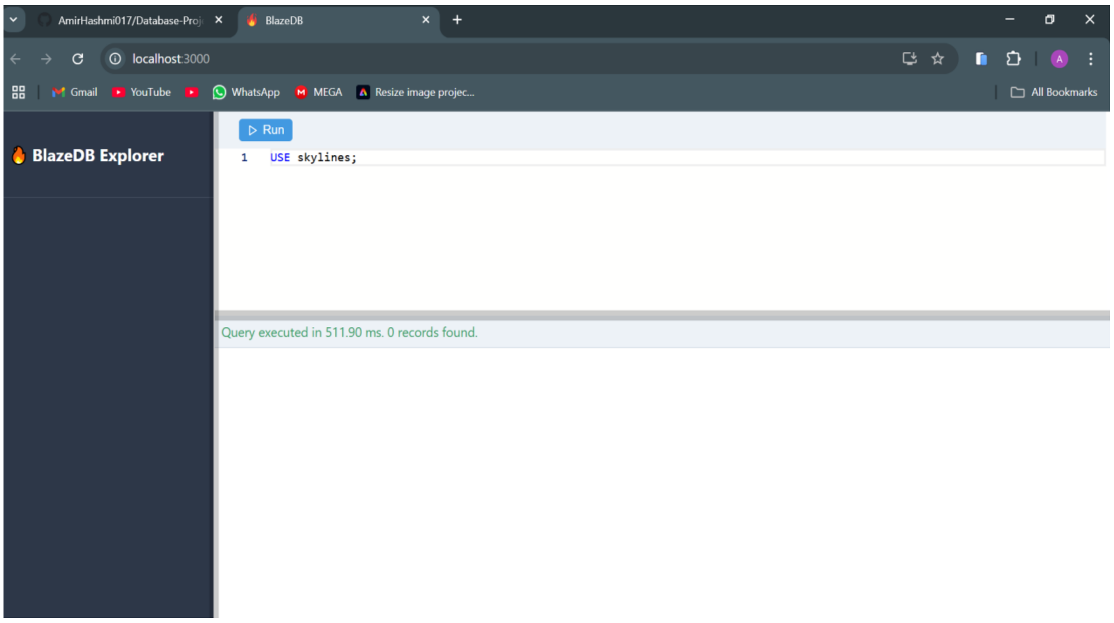


```sql
SELECT users.id,users.name,orders.order_id FROM users JOIN orders ON users.id = orders.user_id;
```

Query executed in 35.20 ms. 1 records found.

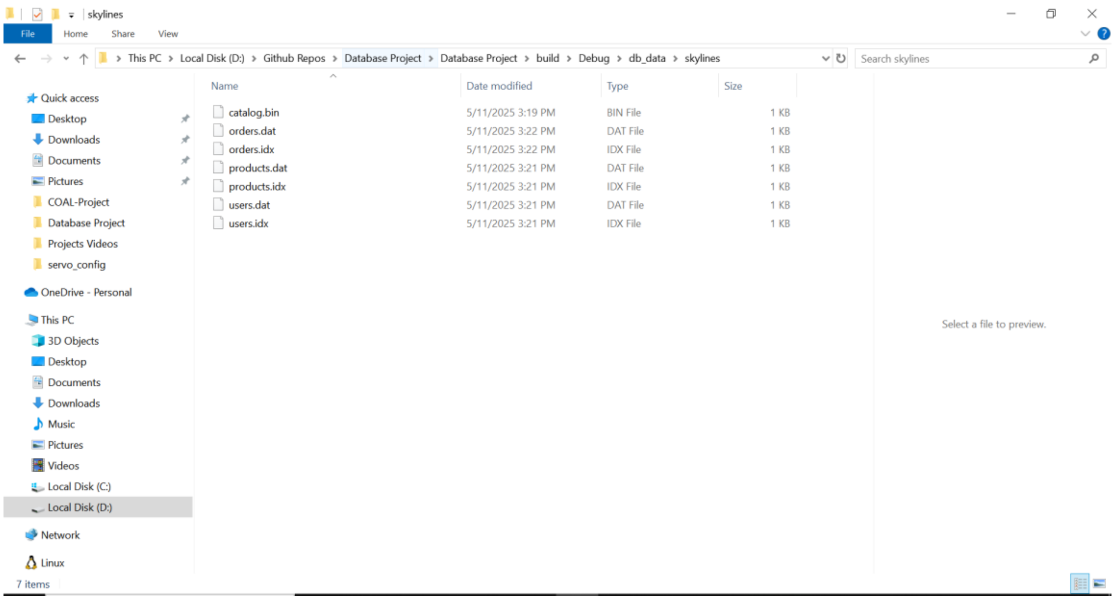
orders.order_id	users.id	users.name
1	1	John Smith

BlazeDB Query Editor



```
USE skylines;
```

```
Query executed in 511.90 ms. 0 records found.
```


skylines
This PC > Local Disk (D:) > Github Repos > Database Project > Database Project > build > Debug > db_data > skylines
7 items

Name	Date modified	Type	Size
catalog.bin	5/11/2025 3:19 PM	BIN File	1 KB
orders.dat	5/11/2025 3:22 PM	DAT File	1 KB
orders.idx	5/11/2025 3:22 PM	IDX File	1 KB
products.dat	5/11/2025 3:21 PM	DAT File	1 KB
products.idx	5/11/2025 3:21 PM	IDX File	1 KB
users.dat	5/11/2025 3:21 PM	DAT File	1 KB
users.idx	5/11/2025 3:21 PM	IDX File	1 KB

Backend Storage

6. B+ Tree Implementation

6.1 Node Structure

```
struct BPlusNode {
    bool is_leaf;
    int parent;
    std::vector<int> keys;
    std::vector<int> children;    // Non-leaf nodes
    std::vector<int> data_ptrs;   // Leaf nodes
};
```

6.2 Operations

Operation	**Complexity**	**Details**
Insert	O(log n)	Split nodes on overflow
Delete	O(log n)	Merge nodes under threshold
Search	O(log n)	Binary search within nodes

7. Query Processing

7.1 Workflow

Query Parser Tokenize Query and call corresponding function from Database Manager.

7.2 Example Queries

7.2.1 Create and Drop Database

```
CREATE DATABASE skylines;

USE skylines;

DROP DATABASE skylines;
```

7.2.2 Create and Drop Table

```
CREATE TABLE users (id INT, name STRING(50), age INT, PRIMARY
KEY (id));

DROP TABLE users;
```

7.2.3 CRUD Operations on Table

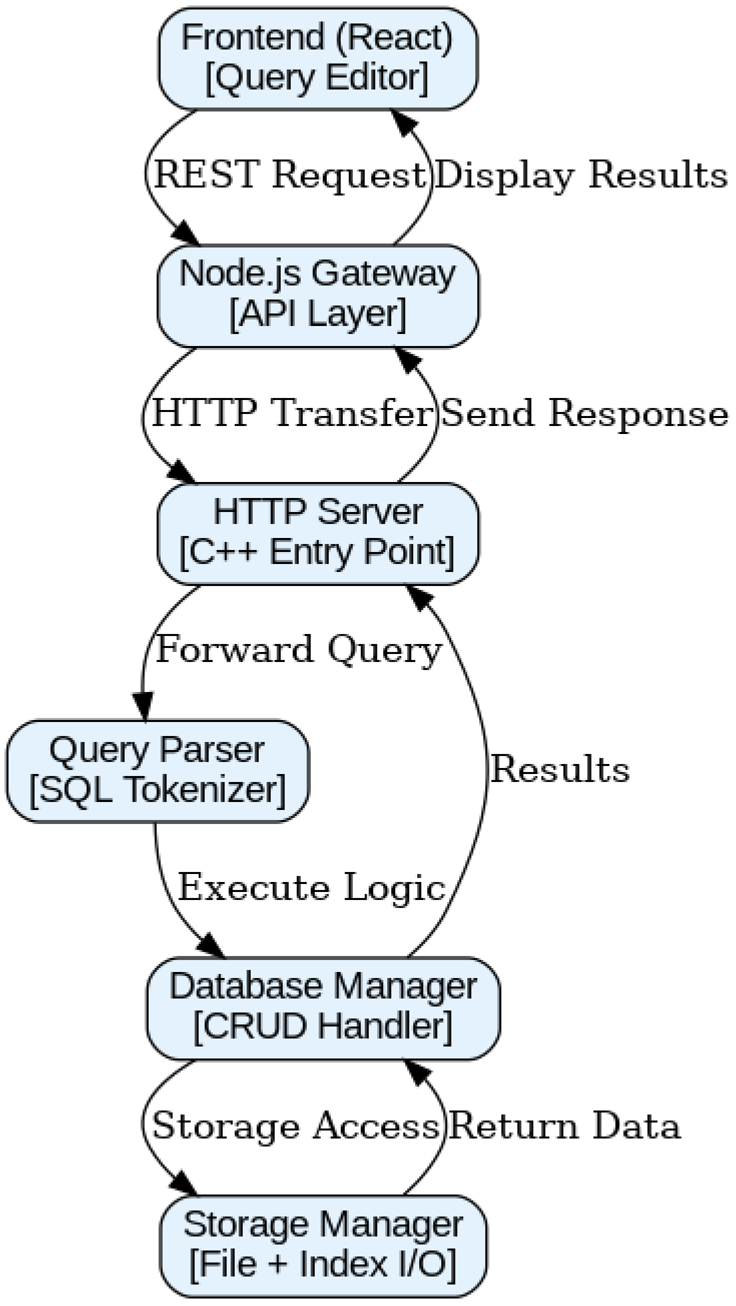
```
INSERT INTO users VALUES (1, 'John Doe', 30);

UPDATE users SET name = 'John Smith' WHERE id = 1;

DELETE FROM users WHERE age < 30;

SELECT users.id,users.name,orders.order_id FROM users JOIN
orders ON users.id = orders.user_id WHERE users.id = 1 and
users.name = ‘Amir’;
```

8. App Flow



9. Installation Guide

9.1 Backend (C++)

9.1.1 Clone the Github Repository:

```
git clone git@github.com:AmirHashmi017/Database-Project.gitcd
Database Project/Database Project
```

9.1.2 Build with CMake:

```
mkdir build

cd build

cmake ..

cmake --build .cd debug

./Database.exe
```

Backend Server is running on port 8080

9.2 Gateway Service Setup

9.2.1 Navigate to the gateway-service directory:

```
cd Database Project/gateway-service
```

9.2.2 Install dependencies & run:

```
npm install

node server.js
```

Gateway Service is running on port 3001

9.3 Frontend Setup

9.3.1 Navigate to the frontend directory:

```
cd Database Project/db-frontend
```

9.3.2 Install dependencies & run:

```
npm install

npm run start
```

Access the app at http://localhost:3000

10. Future Improvements

- **Advanced Data Types:** Support for a wide range of data formats including JSON and Blob data.
- **User Interface:** Intuitive and user-friendly UI with drag-and-drop features and click-based operations for easy navigation and data management.
- **CSV Support:** Seamless import and export of data in CSV format for integration with external tools and workflows.

11. References

Github: https://github.com/AmirHashmi017/Database-Project

Linkedin: Project Working Video