Advance Database Management System

Project Report

Section: A

Submitted by:

Fiza Manzoor	2023-CS-612
Samar Noor	2023-CS-630
Ghulam Fareed	2023-CS-621

Submitted to:

Mam. Rida

Dated: 27th April 2025

Department of Computer Science

University of Engineering and Technology Lahore, New Campus

Database Benchmark Tool Report

The **Database Benchmark Tool** is a Stream lit-based application designed to compare the performance of four database systems—PostgreSQL, SQL Server, MongoDB, and Redis—across various operations such as CRUD, Indexing, Joins, and Hash Queries. The tool provides a user-friendly interface to establish database connections, populate test data, execute benchmarks, visualize results, and clean up data post-testing. This report details each feature of the tool, its implementation, and includes visualizations of benchmark results based on the provided screenshots.

Features and Implementation

1. Database Connection Management

Description: The tool supports connecting to four database systems: PostgreSQL, SQL Server, MongoDB, and Redis. It provides real-time feedback on connection status and allows users to connect or disconnect all databases via the sidebar.

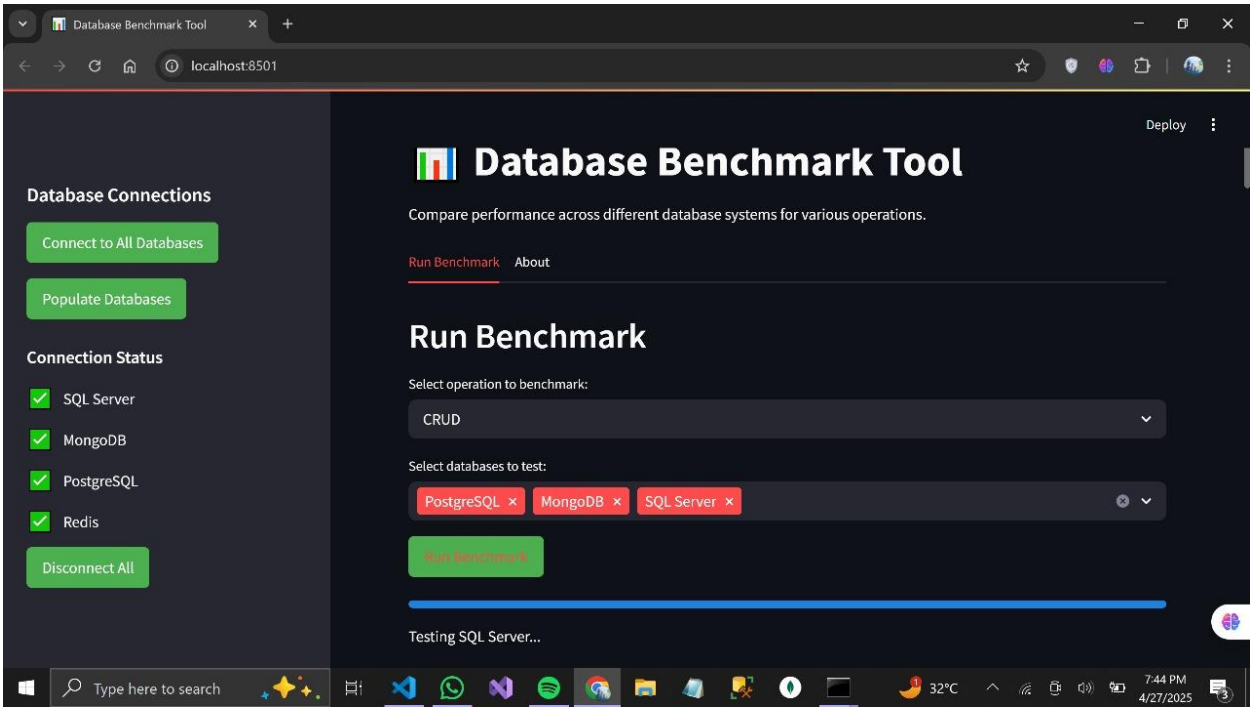
Implementation:

Class: DatabaseConnections

- **Methods**:
  - connect_all(): Establishes connections to all databases concurrently using Stream lit columns for visual feedback.
  - _connect_sql_server(), _connect_mongodb(), _connect_postgresql(), _connect_redis(): Individual connection methods for each database, handling specific connection strings and error cases.
  - close_all(): Closes all active database connections and updates the connection status.
- **Connection Status**: Displayed in the sidebar with icons (✅ for connected, ❌ for disconnected) using custom CSS for styling.
- **Error Handling**: Catches and displays specific exceptions (e.g., pyodbc.Error, ConnectionFailure) for each database.

Usage:

- Users click "Connect to All Databases" to initiate connections.
- Connection status is updated dynamically, showing success or failure for each database.



2. Data Population

Description: The tool populates each database with 10,000 customer records and a variable number of account records (1–3 per customer) to create a realistic data set for bench marking.

Implementation:

- **Function**: populate_data(db_name, conn_obj)
- **Process**:
    - Cleans existing data using cleanup_data() to ensure a fresh dat aset.
    - Generates 10,000 customers with unique IDs, names, and emails.
    - Creates 1–3 accounts per customer with random balances and account types (Savings, Checking, Credit).
    - Inserts data using database-specific queries or commands:
        - **PostgreSQL/SQL Server**: Uses SQL INSERT statements with table creation.
        - **MongoDB**: Uses insert_many() for bulk insertion into Customers and Accounts collections.
        - **Redis**: Uses hset to store customer and account data as hash sets.
- **Error Handling**: Rolls back transactions for SQL databases and displays errors via Streamlit.

Usage:

- Users click "Populate Databases" in the sidebar to generate and insert test data.
- Success messages confirm the number of customers and accounts populated.

3. Data Cleanup

Description: Ensures databases are reset after each benchmark by deleting all populated data and dropping indexes.

Implementation:

- **Function**: cleanup_data(db_name, conn_obj)
- **Process**:
    - **PostgreSQL**: Drops indexes (idx_email, idx_email_hash) and deletes data from Customers and Accounts tables.
    - **SQL Server**: Drops the idx_email index and deletes data from tables.
    - **MongoDB**: Drops Customers and Accounts collections.
    - **Redis**: Flushes the entire database (flushdb).

Usage:

- Automatically triggered after a benchmark run to clean up data and indexes.
- Users receive confirmation of successful cleanup or error messages.

4. Benchmark Execution

Description: The tool executes performance tests for four operation types: CRUD, Indexing, Joins, and Hash Queries, measuring wall time, CPU time, and memory usage.

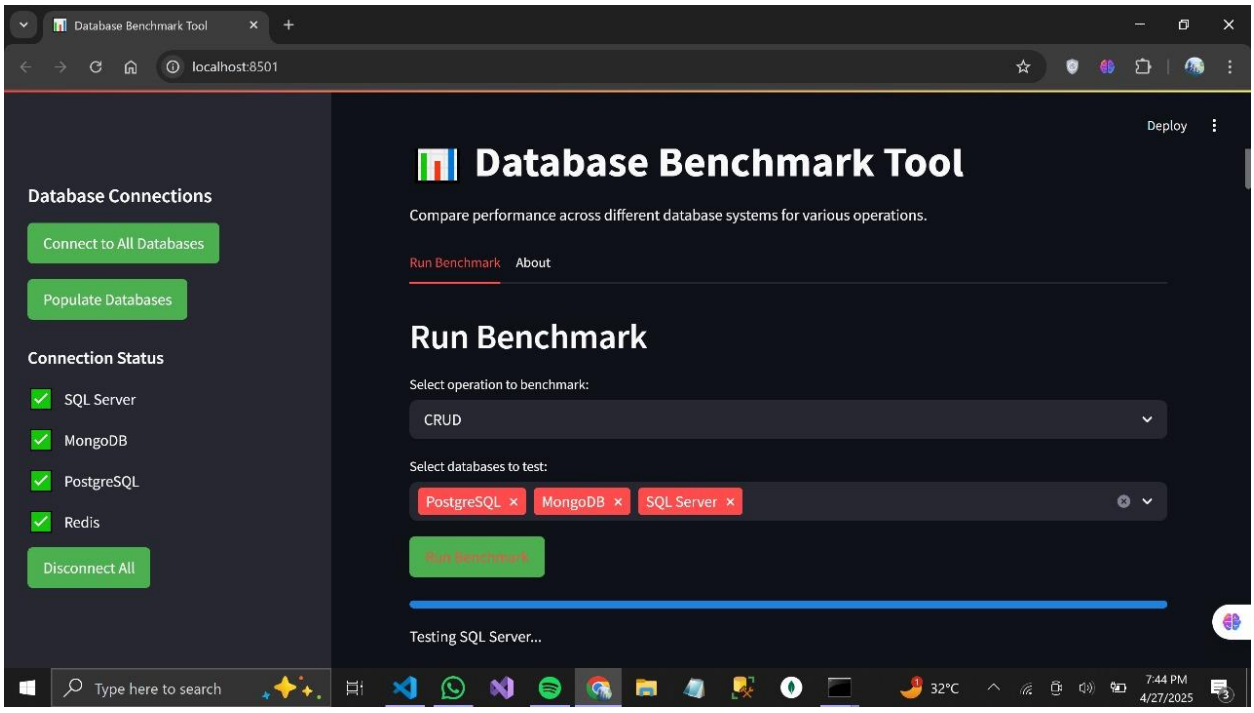
Implementation:

- **Function**: execute_query(db_name, conn_obj, query_type)
- **Operation Types**:
    - **CRUD**: Performs batch operations (100 rows) including insert, select, update, and delete.
    - **Indexing**: Creates an index on the email field and queries for emails starting with 'a'.
    - **Joins**: Joins Customers and Accounts tables/collections, filtering by email (SQL/MongoDB) or simulates joins client-side (Redis).

    - **Hash Queries**: Uses hash-based indexes for fast lookups, querying multiple emails.
- **Metrics**:
    - **Wall Time**: Total execution time in milliseconds.
    - **CPU Time**: CPU time used for the query in milliseconds.
    - **Memory Usage**: Memory consumed in MB, calculated using psutil.
- **Database-Specific Logic**:
    - **PostgreSQL/SQL Server**: Uses SQL queries with cursors and transactions.
    - **MongoDB**: Uses collection operations and aggregation pipelines (for Joins).
    - **Redis**: Uses hash sets and sorted sets for operations, with client-side logic for Joins.

Usage:

- Users select an operation type and databases to test in the "Run Benchmark" tab.
- A progress bar and status text provide real-time feedback during execution.



5. Result Visualization and Reporting

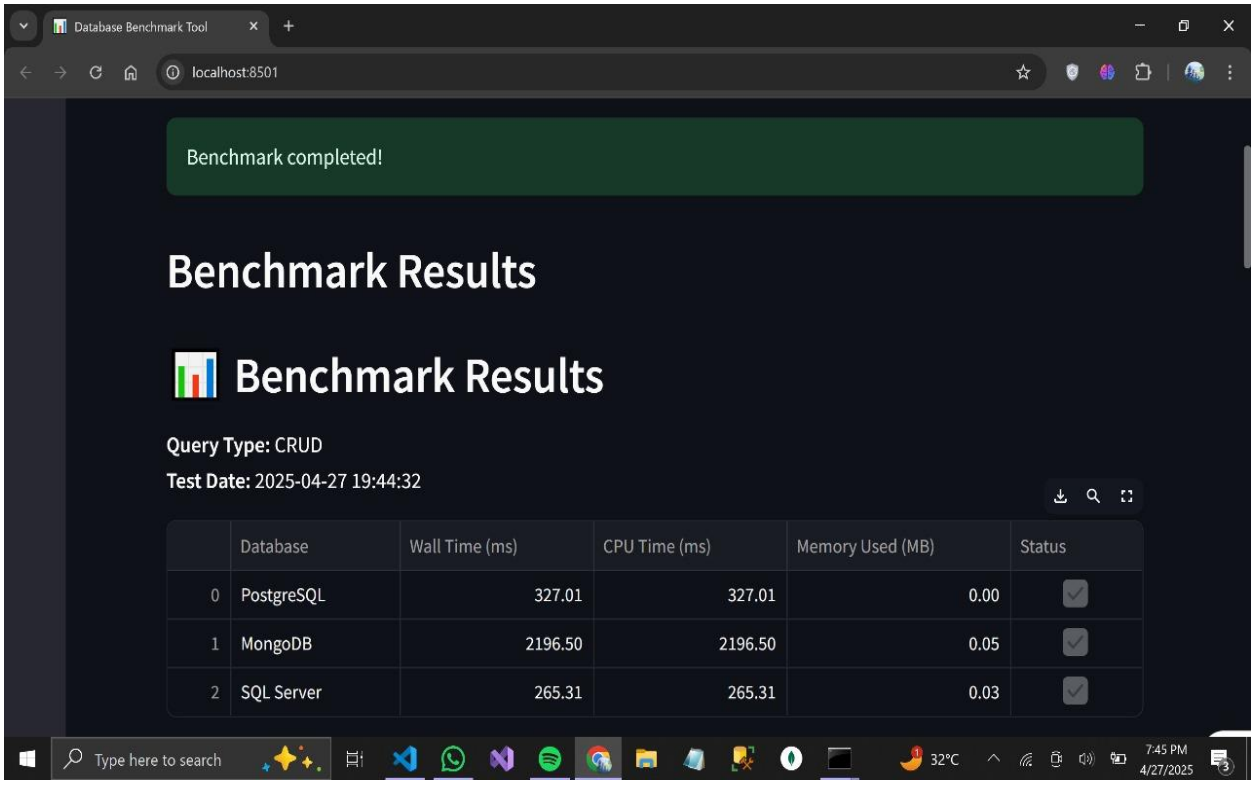
Description: The tool generates a comprehensive report with tabular data, bar charts, and analysis for each benchmark run.

Implementation:

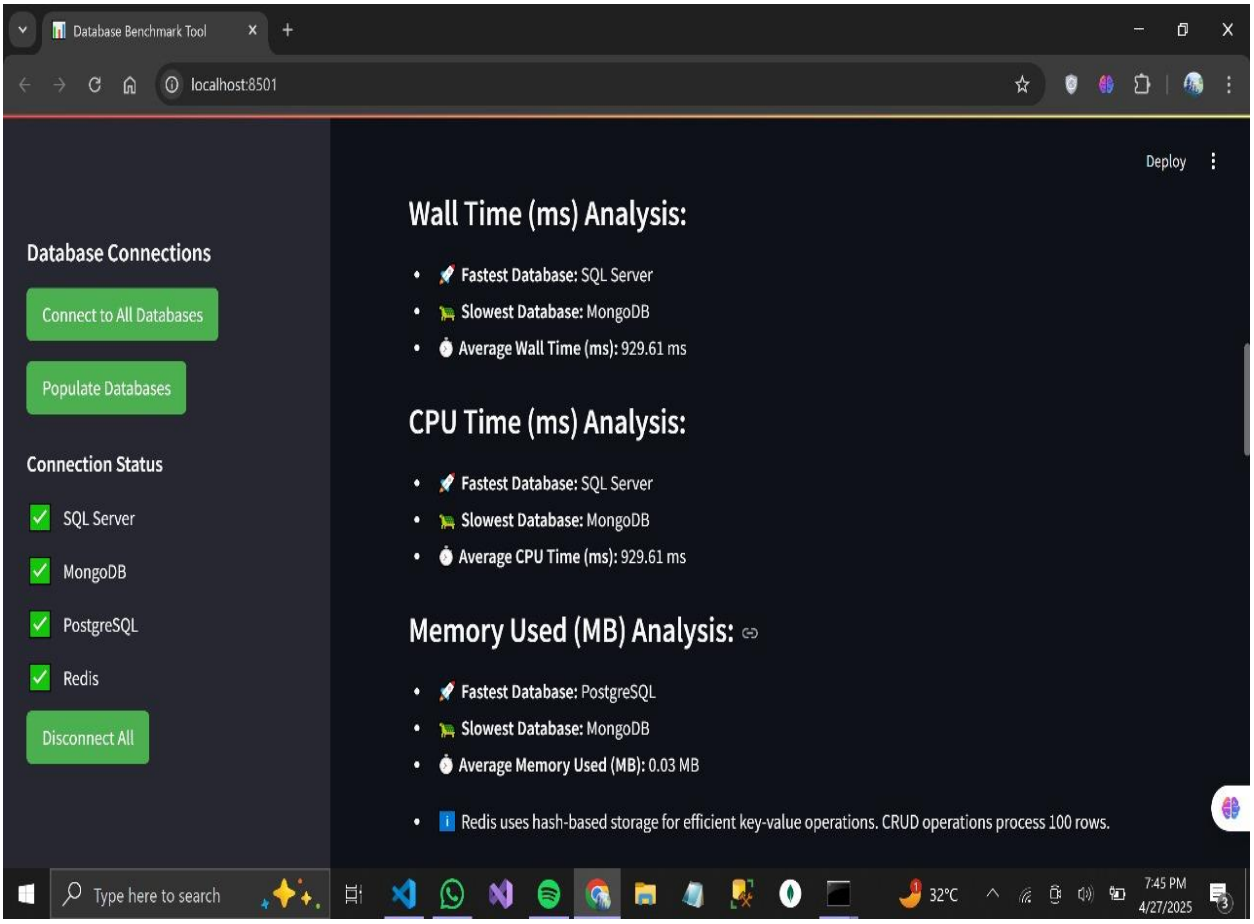
- **Function**: generate_report(results, query_type)
- **Components**:
    - **Data Frame**: Displays results in a table with columns for Database, Wall Time, CPU Time, Memory Used, and Status.
    - **Analysis**:
        - Identifies the fastest and slowest databases for each metric.
        - Calculates average metric values.
        - Provides context-specific notes (e.g., Redis uses client-side joins).
    - **Visualizations**:
        - Bar charts for Wall Time, CPU Time, and Memory Used using Plotly Express.
        - Charts include formatted text labels, custom colors, and a transparent background.
    - **Sample Data**: Displays query results in expandable sections for each database.
- **Styling**: Uses custom CSS for a polished look, with shadows and rounded corners for the report container.

Usage:

- Results are displayed after a benchmark run, with interactive charts and downloadable CSV files.
- Users can expand sample data to inspect query outputs.



- **Wall Time Analysis**:
    - Fastest Database: SQL Server
    - Slowest Database: MongoDB
    - Average Wall Time: 929.61 ms
- **CPU Time Analysis**:
    - Fastest Database: SQL Server
    - Slowest Database: MongoDB
    - Average CPU Time: 929.61 ms
- **Memory Used Analysis**:
    - Fastest Database: PostgreSQL
    - Slowest Database: MongoDB
    - Average Memory Used: 0.03 MB A note at the bottom mentions that Redis uses hash-based storage for efficient key-value operations and that CRUD operations process 100 rows.

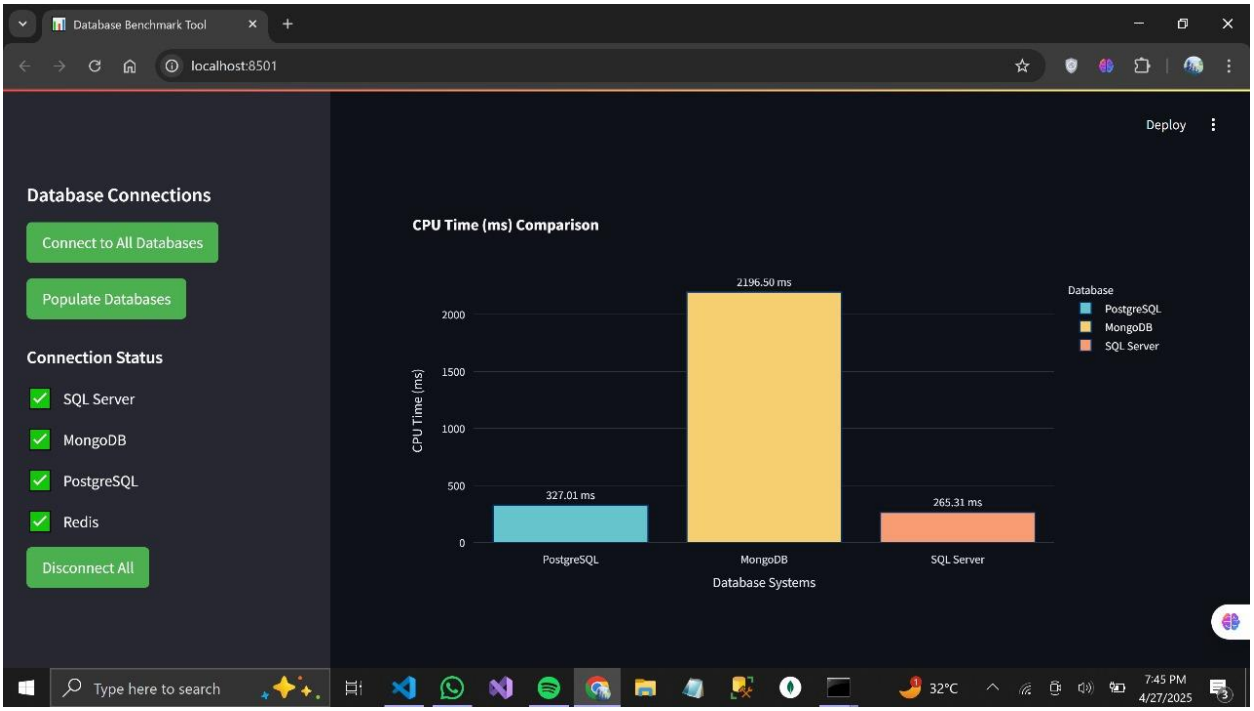


Wall Time Analysis

- **PostgreSQL**: 327.01 ms
- **MongoDB**: 2196.50 ms
- **SQL Server**: 265.31 ms MongoDB has the highest wall time, while SQL Server is the fastest.

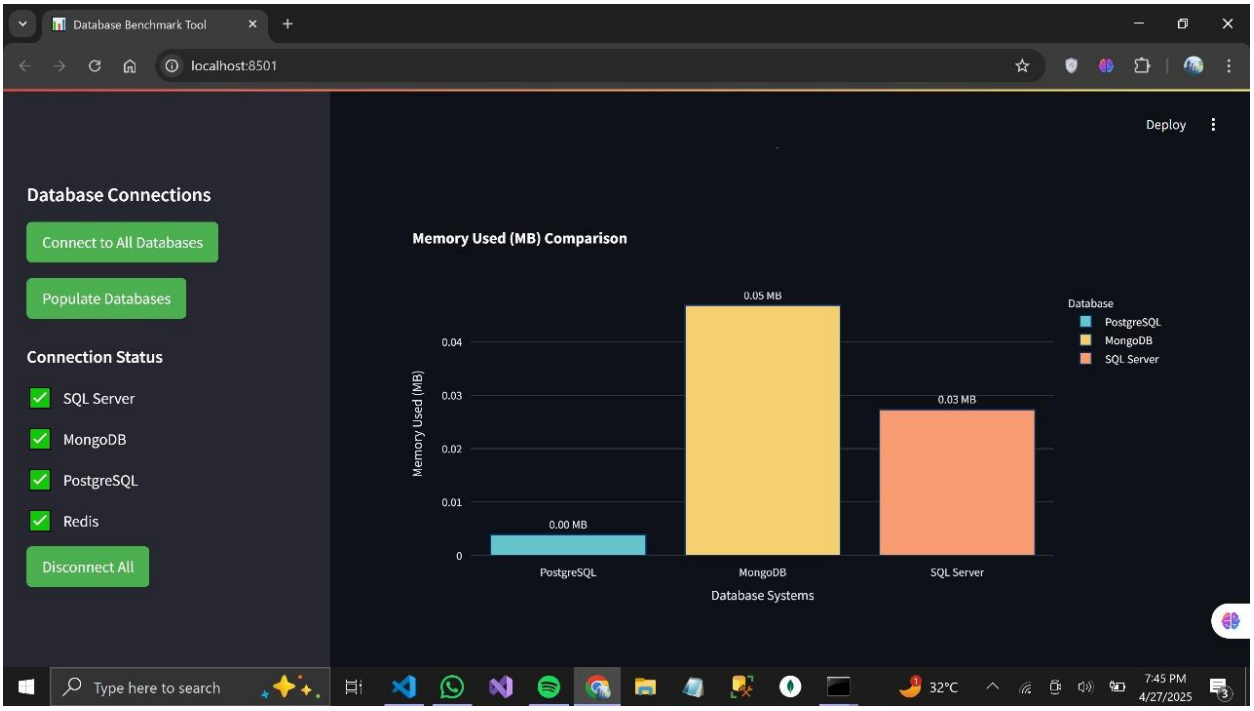
Cpu based Comparison

- **PostgreSQL**: 327.01 ms
- **MongoDB**: 2196.50 ms
- **SQL Server**: 265.31 ms The results mirror the wall time, with MongoDB being the slowest and SQL Server the fastest.



Memory Based Comparison

- **PostgreSQL**: 0.00 MB
- **MongoDB**: 0.05 MB
- **SQL Server**: 0.03 MB PostgreSQL uses the least memory, while MongoDB uses the most.



6. CSV Download

Description: Allows users to download benchmark results as a CSV file for further analysis.

Implementation:

- **Function**: get_download_link(df, filename)
- **Process**:
    - Converts the results Data Frame to CSV format.
    - Encodes the CSV in base64 and generates a download link using HTML.
- **Integration**: The download link is displayed below the benchmark results.

Usage:

- Users click the "Download CSV File" link to save the results, as shown in the "Sample Data from Queries" screenshot.

7. User Interface

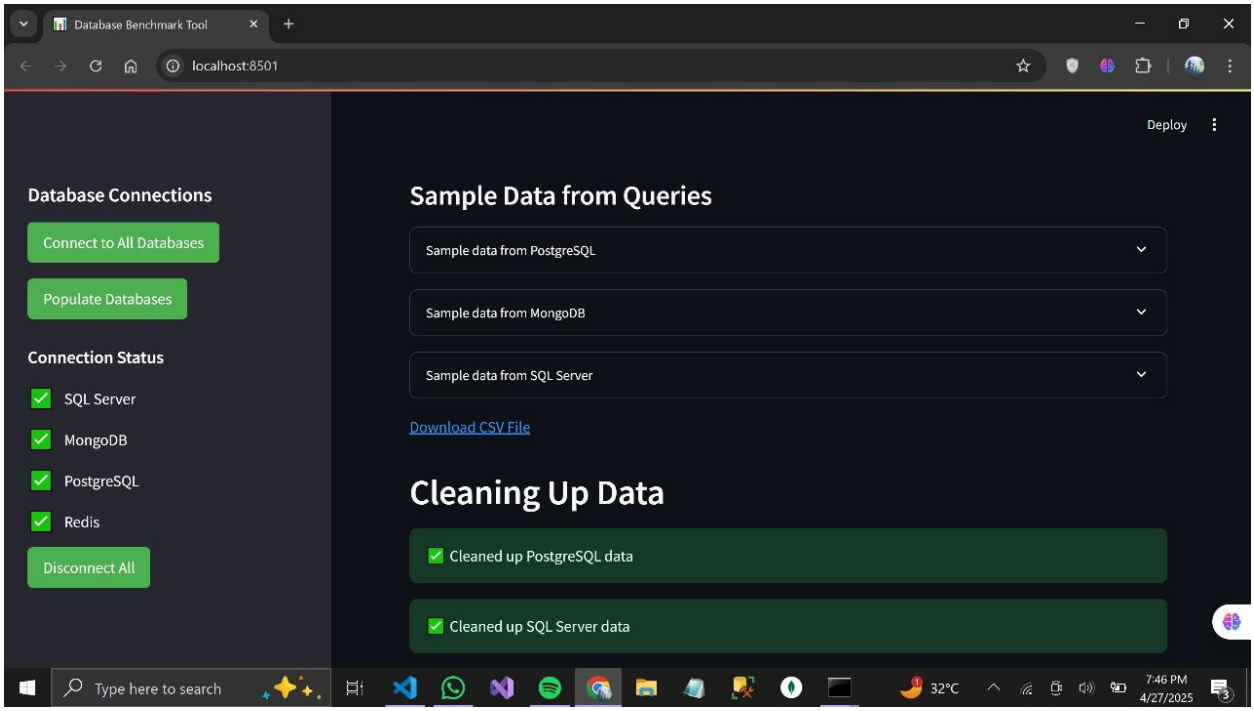
Description: The Stream lit interface is intuitive, with a sidebar for database controls and tabs for bench marking and documentation.

Implementation:

- **Sidebar**:
    - Buttons for connecting, populating, and disconnecting databases.
    - Connection status display with icons.
- **Tabs**:
    - **Run Benchmark**: Allows selection of operation type and databases, displays results.
    - **About**: Provides documentation on the tool's features and usage.
- **Custom CSS**: Enhances the UI with a light background, styled buttons, and status icons.
- **Session State**: Persists database connections and results across interactions.

Usage:

- Users navigate the sidebar to manage databases and the main tabs to run benchmarks or view documentation, as visible in all screenshots.



Technical Details

- **Libraries**:
    - **Database Connectors**: pyodbc (SQL Server), pymongo (MongoDB), psycopg2 (PostgreSQL), redis (Redis).
    - **Data Processing**: pandas for result handling, psutil for memory and CPU metrics.
    - **Visualization**: plotly.express for interactive charts, streamlit for the UI.
    - **Utilities**: random, string, base64, io for data generation and file handling.
- **Performance Considerations**:
    - Batch processing for CRUD operations to reduce overhead.
    - Concurrent connection establishment using Streamlit columns.
    - Efficient memory management with psutil for accurate metrics.

Usage Instructions

1. **Setup**: Ensure PostgreSQL, SQL Server, MongoDB, and Redis are running locally with the configured connection strings.
2. **Connect**: Use the sidebar to connect to all databases.
3. **Populate**: Populate databases with test data (10,000 customers, variable accounts).
4. **Benchmark**: Select an operation type (CRUD, Indexing, Joins, Hash Queries) and databases, then run the benchmark.

5. **Analyze**: Review the generated report, including tables, charts, and sample data.
6. **Download**: Save results as a CSV file.
7. **Cleanup**: Data and indexes are automatically deleted after each run.

Notes

- **Redis Joins**: Implemented client-side due to Redis's key-value nature, which may affect performance comparisons.
- **MongoDB Joins**: Uses aggregation pipelines, which are less performant than SQL joins but more flexible.
- **Hash Queries**: Optimized for fast look ups, with Redis using hash sets and MongoDB using hashed indexes.
- **Scalability**: The tool is designed for 10,000 records but can be modified for larger datasets by adjusting the populate_data function.

Conclusion

The Database Benchmark Tool provides a robust platform for comparing database performance across diverse operations. Its intuitive interface, comprehensive reporting, and automated cleanup make it suitable for developers, database administrators, and researchers evaluating database systems. The visualizations, as shown in the screenshots, enhance result interpretation, while the CSV export feature supports further analysis. The benchmark results highlight SQL Server as the fastest for CRUD operations in terms of wall and CPU time, while PostgreSQL is the most memory-efficient.