**A Project Report for DataBase Management Systems Lab(22CS406PC)**

**On**

**WORKOUT TRACKER**

**Submitted**

**to**

**CMR Technical Campus, Hyderabad**

**In Partial fulfillment for the requirement of the Award of the Degree of**

#### BACHELOR OF TECHNOLOGY

##### in

#### COMPUTER SCIENCE & ENGINEERING

#### by

## MUMMADI ADARSH CHARY

## (237R1A05BF)

**Under the esteemed guidance of**

## Ms. B. Pooja Reddy

## Assistant Professor

****

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CMR TECHNICAL CAMPUS**

***An UGC Autonomous Institute***

**Accredited by NBA & NAAC with A Grade**

**(Approved by AICTE, Affiliated to JNTU, Hyderabad)**

**Kandlakoya (V), Medchal (M), Hyderabad-501 401**

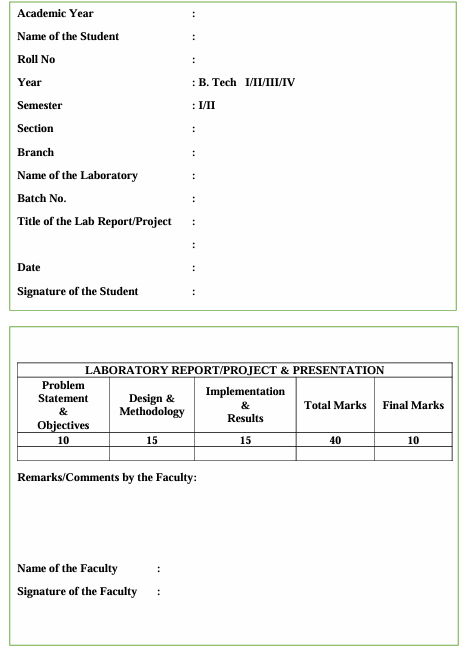
**(2024-2025)**



**CERTIFICATE**

This to certify that, the Presentation entitled **“WORKOUT TRACKER”** is submitted by **MUMMADI ADARSH CHARY** bearing the Roll Number **237R1A05BF** of **B. Tech Computer Science and Engineering**, In Partial fulfillment for the requirement of the Presentation and for the award of the **Degree of Bachelor of Technology** during the academic year 2024-25.

**Subject Faculty**

****

|  |
| --- |
| **VISION** |
| * To produce technically proficient and skilled graduates in information technology and to * prepare them to meet the needs of industry and society in a holistic learning environment |
| **MISSION** |
| * To prepare students with the essential knowledge and contemporary abilities for solving * emerging technological problems. * To train the students on cutting – edge computing facilities with industry interaction to * enhance in practical exposure to real world situations. * To enhance the leadership qualities and team spirit with social values to serve the society. |

**INSTITUTION VISION, MISSION**

|  |
| --- |
| **VISION** |
| To Provide quality education and a conducive learning environment in computer engineering  that foster critical thinking, creativity, and practical problem-solving skills. |

|  |
| --- |
| **MISSION** |
|  |
| 1. To educate the students in fundamental principles of computing and induce the skills needed 2. to solve practical problems. 3. To provide State-of-the-art computing laboratory facilities to promote industry-institute 4. interaction to enhance student’s practical knowledge. 5. To Inculcate self-learning abilities, team spirit, and professional ethics among the students 6. to serve society. |

**TABLE OF CONTENTS:**

1. ABSTRACT 6
2. INTRODUCTION 7
3. PURPOSE 8 - 10
4. OBJECTIVES 11 - 12
5. USER INSTRUCTIONS 13 - 17
6. SAMPLE CODE 17
7. OUTPUT 21 - 22
8. FUTURE IMPROVEMENTS 23 - 24
9. CONCLUSION 25

10.REFERENCES 26

1. **ABSTRACT**

WorkoutTracker is a full-stack fitness logging web application built using Node.js, Express, EJS, and MySQL. It provides an interactive platform for users to log and track workouts like exercises, sets, reps, and weight. The project places a strong focus on SQL queries for data manipulation, user management, and workout visualization. Key SQL operations include INSERT, SELECT, UPDATE, DELETE, and JOIN queries to manage and analyze user workout data. The project showcases real-world application of SQL in backend logic, form handling, and data visualization using Chart.js.

**2.INTRODUCTION**

The WorkoutTracker project is a full-stack fitness tracking application developed as part of a Database Management Systems (DBMS) lab. It provides users with a platform to record, update, and monitor their workouts digitally. The core of the project lies in its efficient and structured use of SQL to manage user and workout data, making it a practical demonstration of real-world database application.

This system utilizes MySQL, a relational database, to store user credentials and workout logs. Each user’s workout data is tied through a foreign key (user\_id), enabling the system to maintain data consistency and referential integrity. The database schema is designed in a normalized format to avoid redundancy and ensure scalability as user data grows over time.

The backend, powered by Node.js and Express, interacts with the database using SQL commands for all CRUD operations. When a user logs a new workout, the application executes an INSERT query to store the record. The home page displays recent workouts and total sessions using SELECT queries with filters and ordering. The history section uses advanced queries to fetch all workouts in descending order, while edits and deletions are handled via UPDATE and DELETE statements, respectively.

Authentication is secured using SQL-based validation, where user data is matched during login through:

SELECT \* FROM users WHERE username = ?;

To enhance user experience, Chart.js is integrated to visualize progress over time. These charts are populated using data from SQL queries that fetch recent exercise weights and reps.

In summary, the WorkoutTracker project demonstrates how a web application can be efficiently driven by a structured SQL backend. It showcases user authentication, workout logging, inline editing, real-time statistics, and charting — all powered by carefully crafted SQL queries. This integration of database systems with web technologies highlights the practical value of SQL in modern application development.

**3.PURPOSE**

The WorkoutTracker project has been developed to demonstrate how database systems, particularly SQL-based relational databases, can be integrated into real-world web applications for structured and secure data handling. The project focuses on fitness logging, where users can log their workouts, analyze their performance, and view visual representations of progress over time. The primary purpose of this project is to showcase the effective use of SQL in user management, data manipulation, analytics, and session handling.

🟢 1. To Develop a Data-Driven Fitness Logging Application

One of the primary goals of this project is to create an interactive and user-friendly platform for logging and reviewing workout sessions. Users can input data such as the workout name, exercise performed, number of sets, repetitions, and weight lifted. This data is then stored in a MySQL database, where it is managed and retrieved using SQL queries. Each user has access to their own records, thanks to the structured relational schema and foreign key relationships that link workouts to users.

🟢 2. To Leverage SQL for Secure User Authentication

The project includes user sign-up, login, and logout functionality. All user details — including usernames, emails, and encrypted passwords — are stored in the users table. SQL queries are used to check whether a username already exists or to verify user credentials during login.

SELECT \* FROM users WHERE username = ?;

Passwords are hashed before being stored, and token-based authentication ensures that only verified users can perform data operations. SQL plays a crucial role in checking user existence and controlling access based on stored credentials.

🟢 3. To Enable Real-Time Workout Tracking Using SQL Queries

Each workout entry by the user is inserted into the workoutData table using:

INSERT INTO workoutData (Workoutname, workout\_date, Exercise\_name, t\_sets, reps, weight, user\_id)

VALUES (?, ?, ?, ?, ?, ?, ?);

This data is later retrieved to show recent workouts, workout statistics, and chart-based progress reports. The use of SELECT queries with filters such as ORDER BY, LIMIT, and DISTINCT allows for personalized and optimized data retrieval.

🟢 4. To Demonstrate CRUD Operations with SQL

The core operations of the application revolve around Create, Read, Update, and Delete (CRUD):

* Create: Adding new workout entries using INSERT
* Read: Displaying user-specific logs using SELECT
* Update: Modifying records using UPDATE
* Delete: Removing workouts individually or in bulk using DELETE

Each of these is performed through server-side logic that communicates with the database using well-structured SQL queries.

🟢 5. To Provide Visual Analytics Backed by SQL Data

To make fitness tracking more insightful, the project uses Chart.js to generate line graphs that reflect performance over time. These graphs are powered by SQL queries that fetch the last 10 records of a specific exercise:

SELECT workout\_date, weight, reps FROM workoutData WHERE user\_id = ? AND Exercise\_name = ?

ORDER BY workout\_date DESC

LIMIT 10;

By processing this SQL data, the frontend can generate meaningful charts that help users visualize their strength progression, consistency, and trends in performance.

🟢 6. To Ensure Data Integrity and Relational Consistency

A key purpose of using MySQL is to enforce relational constraints that maintain data integrity. The user\_id field in the workoutData table references the id field in the users table. This foreign key ensures that each workout is always associated with a valid user, and deletion of users can be prevented or cascaded based on rules.

🟢 7. To Simulate a Real-World DBMS Use Case

This project mirrors the behavior of real-world DBMS applications — including authentication, validation, session-based access, relational joins, filtered search, and performance analytics — all managed via SQL. By doing so, it demonstrates how database theory translates into practical software design using MySQL as the backend.

🟢 8. To Provide a Scalable and Extendable SQL-Based Design

The current database schema is normalized and optimized for extension. Future enhancements such as workout categories, user roles (admin/trainer), and comments can be easily added with minimal changes to the schema. This flexibility is achieved through thoughtful SQL design, use of AUTO\_INCREMENT, and logical field grouping.

**4.OBJECTIVES**

The primary objective of the **WorkoutTracker** project is to implement a web-based fitness tracking system where all data operations are managed using **Structured Query Language (SQL)**. The project aims to demonstrate how SQL can be used effectively to perform data storage, retrieval, manipulation, and visualization in real-world applications. The system leverages **MySQL** as the backend database and focuses on using optimized SQL queries for each feature of the application.

**🔹 1. To Design a Relational Database Schema**

* Create normalized tables (users, workoutData) that represent users and their workout logs.
* Use **primary keys** for unique identification and **foreign keys** to maintain relationships.

**🔹 2. To Perform CRUD Operations Using SQL**

* Enable users to **create** workout logs (INSERT), **read** data (SELECT), **update** existing entries (UPDATE), and **delete** workouts (DELETE).
* Implement these operations securely and efficiently via Express routes.

**🔹 3. To Enable User Authentication and Session Handling**

* Use SQL queries to verify user credentials and retrieve user data during login.
* Example:

SELECT \* FROM users WHERE username = ?;

**🔹 4. To Retrieve and Analyze Data Using SQL**

* Use aggregation queries to count total workout sessions:

SELECT COUNT(\*) FROM workoutData WHERE user\_id = ?;

* Use ORDER BY, DISTINCT, and LIMIT clauses to filter and sort results dynamically.

**🔹 5. To Integrate SQL Data with Frontend Templates**

* Fetch SQL results and embed them in **EJS templates** to render charts, history tables, and workout summaries.

**🔹 6. To Maintain Data Integrity and Security**

* Apply SQL constraints and foreign key relationships to ensure valid data entries.
* Use parameterized queries to prevent SQL injection.

**🔹 7. To Demonstrate Real-World DBMS Application**

* Simulate a scalable, secure, and efficient SQL-driven system that can be extended for future enterprise-level fitness applications.

**5.USER INSTRUCTIONS**

**1. Registration (Sign Up)**

The user begins by registering for the platform through the Sign-Up page. The form collects the following details:

* Username
* Email
* Password

Upon form submission, the password is hashed using bcrypt for security. The following SQL command is used to insert user data into the database:

INSERT INTO users (username, email, password)

VALUES (?, ?, ?);

If the registration is successful, the user is redirected to the login page.

🔹 2. Login

The user can log in using their registered username and password. The backend uses the following SQL command to validate credentials:

SELECT \* FROM users WHERE username = ?;

If a match is found, the hashed password is compared. On success, a session token is created using JWT and stored as a cookie. This token is used to protect all routes that require user authentication.

🔹 3. Dashboard (Home Page)

After logging in, the user is taken to the Dashboard, which shows:

* Total workout sessions (via COUNT)
* A form to log a new workout
* A dropdown to visualize progress using Chart.js
* Recent 3 workouts

SQL commands used:

SELECT COUNT(\*) AS total\_sessions FROM workoutData WHERE user\_id = ?;

SELECT DISTINCT Exercise\_name FROM workoutData WHERE user\_id = ?;

SELECT \* FROM workoutData WHERE user\_id = ? ORDER BY workout\_date DESC LIMIT 3;

🔹 4. Logging a New Workout

The user fills in a form with:

* Workout name
* Exercise name
* Number of sets
* Max reps
* Max weight
* Date

Upon submission, the following SQL command is executed:

INSERT INTO workoutData (Workoutname, workout\_date, Exercise\_name, t\_sets, reps, weight, user\_id)

VALUES (?, ?, ?, ?, ?, ?, ?);

The form clears and the dashboard updates to reflect the new data.

🔹 5. Viewing Workout History

The History page shows all workouts performed by the user. Each row includes:

* Workout name
* Date
* Exercise
* Sets, Reps, Max Weight
* Edit and Delete buttons

SQL command used:

SELECT \* FROM workoutData WHERE user\_id = ? ORDER BY workout\_date DESC;

This allows users to scroll through a full list of past workouts.

🔹 6. Editing a Workout Entry

Each row in the History table includes an Edit button. When clicked:

* The row switches to input fields
* The user updates values
* On saving, a PATCH request is sent to the server

The corresponding SQL UPDATE command:

UPDATE workoutData

SET Workoutname = ?, workout\_date = ?, Exercise\_name = ?, t\_sets = ?, reps = ?, weight = ?

WHERE id = ?;

🔹 7. Deleting a Workout

To delete a single entry, the user clicks the Delete button. This triggers the following SQL command:

DELETE FROM workoutData WHERE id = ?;

There is also a Delete All option which removes all records for the user:

DELETE FROM workoutData WHERE user\_id = ?;

🔹 8. Visualizing Progress (Chart.js)

The user can select an exercise from the dropdown and click “Show Chart”. The backend executes this SQL query:

SELECT workout\_date, weight, reps

FROM workoutData

WHERE user\_id = ? AND Exercise\_name = ?

ORDER BY workout\_date DESC LIMIT 10;

This data is then passed to Chart.js to display progress over time.

🔹 9. Logout

Clicking the Logout button triggers a POST request that clears the user’s session cookie and redirects them to the login page. No SQL command is executed here; it's a cookie operation.

**6.SAMPLE CODE**

This section presents the SQL-focused code snippets and backend integration logic used in the WorkoutTracker application. These examples highlight key database operations such as creation, insertion, retrieval, updating, and deletion of records in a relational schema.

**🔹 1. Creating the Users Table**

The users table stores registered users along with their login credentials (hashed passwords).

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL,

password VARCHAR(255) NOT NULL

);

* AUTO\_INCREMENT ensures each user gets a unique ID.
* password stores hashed passwords using bcrypt before insertion.

**🔹 2. Creating the Workout Table**

The workoutData table stores all workouts logged by users.

CREATE TABLE workoutData (

id INT AUTO\_INCREMENT PRIMARY KEY,

Workoutname VARCHAR(100),

workout\_date DATE,

Exercise\_name VARCHAR(100),

t\_sets INT,

reps INT,

weight INT,

user\_id INT,

FOREIGN KEY (user\_id) REFERENCES users(id)

);

* This structure supports **one-to-many** relationship: one user can log many workouts.
* FOREIGN KEY ensures data integrity between users and their workouts.

**🔹 3. Inserting User During Signup (Node.js → SQL)**

const insertQuery = "INSERT INTO users (username, email, password) VALUES (?, ?, ?)";

connection.query(insertQuery, [username, email, hashedPassword], callback);

**Purpose:** Stores the new user data securely into the users table.

**🔹 4. Validating User During Login**

SELECT \* FROM users WHERE username = ?;

* Used to fetch the user record and compare the hashed password using bcrypt.
* Ensures only registered users can access their workout data.

**🔹 5. Inserting a Workout Entry**

INSERT INTO workoutData

(Workoutname, workout\_date, Exercise\_name, t\_sets, reps, weight, user\_id)

VALUES (?, ?, ?, ?, ?, ?, ?);

* This query is triggered when a user submits the workout form.
* All workouts are tied to a user using user\_id.

**🔹 6. Retrieving Recent Workouts (Home Page)**

SELECT \* FROM workoutData

WHERE user\_id = ?

ORDER BY workout\_date DESC, Workoutname DESC

LIMIT 3;

* Shows the 3 most recent workouts on the home dashboard.
* Uses ORDER BY and LIMIT to sort and restrict results.

**🔹 7. Displaying All Workouts (History Page)**

SELECT \* FROM workoutData

WHERE user\_id = ?

ORDER BY workout\_date DESC;

* Used to show the complete history of the user’s workouts.
* Returns full records for inline editing or deletion.

**🔹 8. Editing a Workout Record**

UPDATE workoutData

SET Workoutname = ?, workout\_date = ?, Exercise\_name = ?,

t\_sets = ?, reps = ?, weight = ?

WHERE id = ?;

* This query updates a specific workout based on the primary key (id).
* Executed via PATCH request using fetch API.

**🔹 9. Deleting a Specific Workout**

DELETE FROM workoutData WHERE id = ?;

* Removes a workout from the database using its unique id.

**🔹 10. Deleting All Workouts for a User**

DELETE FROM workoutData WHERE user\_id = ?;

* Useful for clearing the user's entire history in one action.

**🔹 11. Getting Data for Charts (Last 10 Entries)**

SELECT workout\_date, weight, reps

FROM workoutData

WHERE user\_id = ? AND Exercise\_name = ?

ORDER BY workout\_date DESC

LIMIT 10;

* This query supports the **Chart.js** visualization feature.
* Returns data that is plotted on the front end.

**🔹 12. Counting Total Workout Sessions**

SELECT COUNT(\*) AS total\_sessions

FROM workoutData

WHERE user\_id = ?;

* Used on the dashboard to show the number of sessions logged.

**🔹 13. Fetching Unique Exercises (for Chart Dropdown)**

SELECT DISTINCT Exercise\_name

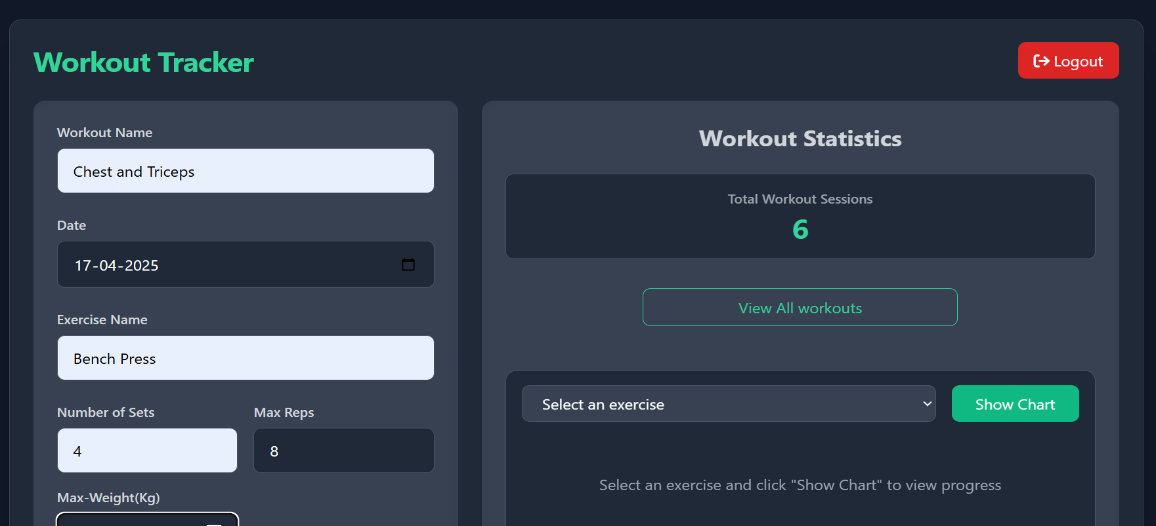
FROM workoutData

WHERE user\_id = ?;

* Retrieves only the unique exercises a user has logged.

**7.OUTPUT**

Main Page **:**



Recent workouts :

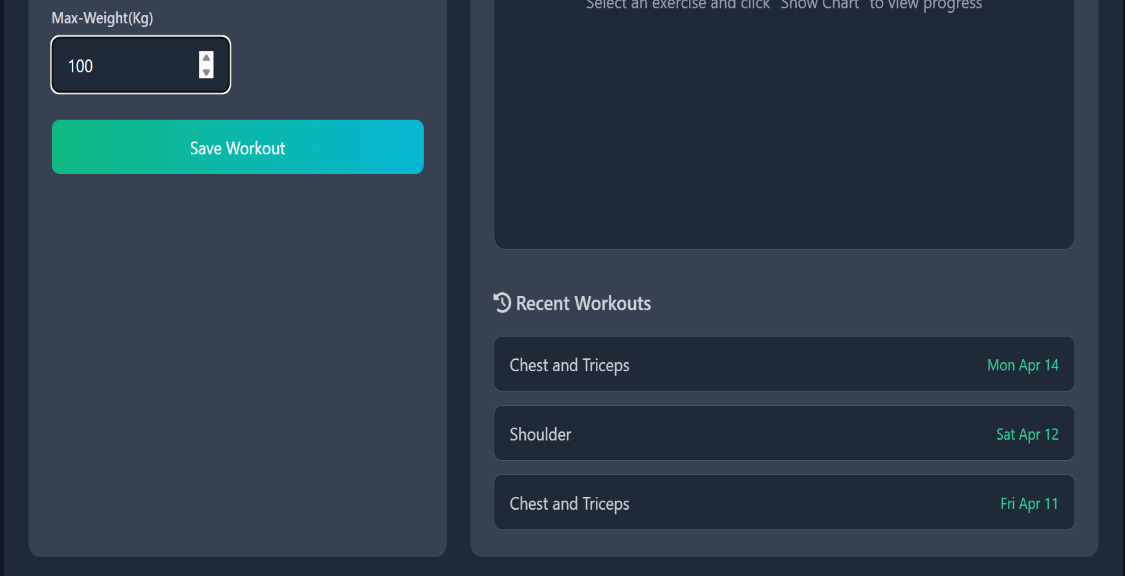
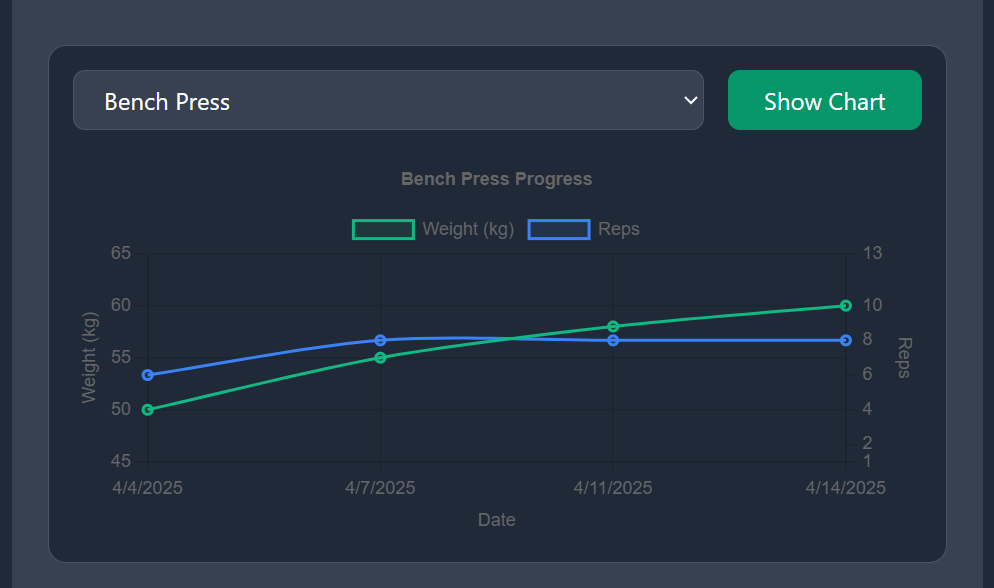
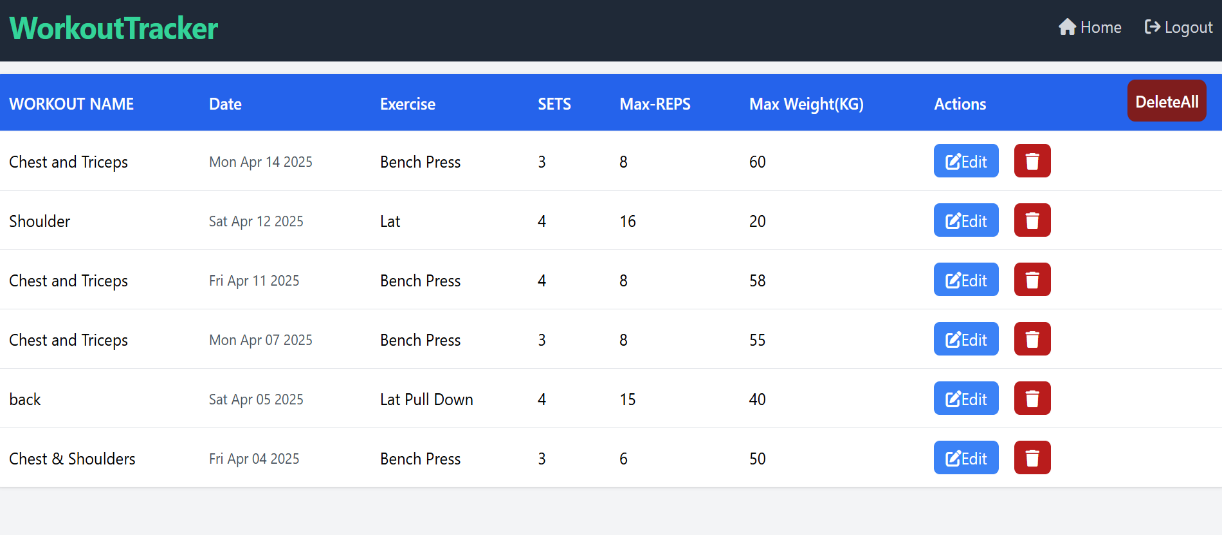


Chart Visualization :



Workout History :



**8.FUTURE IMPROVEMENTS**

The current version of WorkoutTracker provides a solid SQL-based fitness tracking solution. However, several enhancements can be made to improve scalability, performance, and user experience.

🔹 1. Migration from MySQL to PostgreSQL

To improve performance, security, and scalability, the backend database can be migrated from MySQL to PostgreSQL, which offers:

* Advanced data types (JSONB, arrays)
* Full-text search
* Better concurrency handling with MVCC
* Richer indexing and query optimization

This shift would allow more complex queries and better support for analytical features in the future.

🔹 2. Frontend Conversion to React.js

Currently, the UI is rendered using EJS templates. Moving to a React.js-based frontend will:

* Improve responsiveness and performance
* Enable real-time updates using component state
* Allow smoother user interactions and modular development

APIs will continue to communicate with the backend SQL layer for data transactions.

🔹 3. Role-Based Access Control

Introduce roles like admin, trainer, and user using a role column in the users table. SQL-based filtering can enforce permissions and access levels.

🔹 4. Activity Reports and Email Notifications

Generate weekly workout summaries using SQL aggregate queries and send them via scheduled tasks.

🔹 5. Cloud Deployment with Remote SQL

Host the PostgreSQL database on cloud platforms like AWS RDS or Supabase for real-time remote access and backups.

**9.CONCLUSION**

The **WorkoutTracker** project effectively demonstrates the use of SQL in a real-world web application, providing a solid foundation for both the front-end and back-end integration. The primary objective of the project was to design and implement a **relational database** system that efficiently manages user workout data, and SQL has been central in achieving this.

Key SQL-related components of the project include:

1. **Database Design and Schema:**  
   The project’s database schema has been carefully structured to ensure efficient data storage and retrieval. Tables such as **users**, **workouts**, **exercises**, and **workout\_details** are designed to minimize redundancy and maintain data integrity using **foreign keys** and **relationships**.
2. **CRUD Operations:**  
   Through **SQL queries**, we have implemented the basic **Create, Read, Update, Delete (CRUD)** functionality to allow users to log, modify, and remove workout records. These operations ensure that users can interact with their data seamlessly and efficiently.
3. **Complex Queries for Data Retrieval:**  
   Advanced **SQL queries** have been written to support filtering and sorting of workout records based on various parameters like exercise type, date, and maximum weight. This enables users to analyze their progress over time.
4. **Data Integrity & Constraints:**  
   To ensure the accuracy and consistency of data, constraints like **NOT NULL**, **UNIQUE**, and **CHECK** were applied to key fields in the database schema. This guarantees that invalid or incomplete data does not compromise the system's reliability.
5. **Optimized Performance:**  
   By indexing frequently queried fields (like user IDs and workout dates), the application ensures fast and efficient retrieval of data, even as the database grows.
6. **Future Enhancements:**
   * As the project transitions to using **JWT authentication**, SQL queries will be adapted to handle **user authentication** and **authorization** securely.
   * The database will eventually be migrated to a **cloud-based platform**, and SQL queries will be optimized further to handle scalability.

This project not only highlights the importance of SQL in managing dynamic user data but also shows the ability to integrate SQL with other technologies like **Node.js**, **Express**, and **Chart.js** for data visualization.

**10.REFERENCES**

 **Node.js Documentation**

* Node.js Foundation. (2023). *Node.js Documentation*. Retrieved from https://nodejs.org/en/docs/

 **Express Documentation**

* Express.js. (2023). *Express Documentation*. Retrieved from <https://expressjs.com/>

 **Chart.js Documentation**

* Chart.js. (2023). *Chart.js Documentation*. Retrieved from https://www.chartjs.org/docs/latest/

 **JWT Authentication**

* Auth0. (2023). *Introduction to JWT*. Retrieved from

https://auth0.com/learn/json-web-tokens/

[Shradha khapra/Apna College]. (2025). *MySQL Database Lecture*.

[YouTube Channel: Apna College]. Retrieved from [https://youtu.be/hlGoQC332VM?si=Pl\_lL9o9lpcEuBmX]