| Exp: 1 | Problem Identification |
|---|---|

**Aim:** To identify team's project title and its problem statement.

**Project title: Fitness Tracker**

**Problem Statement:**

Our fitness tracking project incorporates a robust backend database, complemented by a user-friendly website that serves as a centralized hub for personalized health and fitness information. Leveraging data analytics, the system tailors exercise recommendations based on an individual's age and target body part. Users can access a curated list of exercises specifically tailored to their needs, fostering a more targeted and efficient approach to fitness routines. The website's intuitive interface not only displays recommended exercises but also allows users to track their progress over time, creating a dynamic and engaging platform that promotes adherence to fitness goals. With a focus on customization and accessibility, our project aims to enhance the user experience in navigating their fitness journey.

**Result:** Project title and problem statement is identified

| Exp: 2 | Requirement Gathering |
|---|---|

**Aim:** To work on collecting project requirements.

**REQUIREMENT GATHERING**: Requirement gathering for a website for a fitness tracker involves understanding the specific needs and functionalities that the website should provide to effectively manage various aspects of the tracker's operations.

**Entities**:

1. **User**:
   - UserID (Primary Key)
   - Gender
   - Height
   - Weight

2. **Activity**:
   - UserID (Foreign Key referencing User.UserID)
   - Date
   - Duration

3. **Workout**:
   - WorkoutID (Primary Key)
   - UserID (Foreign Key referencing User.UserID)
   - Date
   - TotalDuration
   - TotalCaloriesBurned

4. **Account**:
   - AccountID (Primary Key)
   - UserID (Foreign Key referencing User.UserID)
   - PlanType (e.g., free, premium)
   - SubscriptionStartDate
   - SubscriptionEndDate

5.**ExerciseSession**:
   - SessionID (Primary Key)
   - UserID (Foreign Key referencing User.UserID)
   - ActivityID (Foreign Key referencing Activity.ActivityID)
   - StartTime
   - EndTime
   - CaloriesBurned
   - DistanceCovered

6. **ExerciseHistory**:
   - HistoryID (Primary Key)

- UserID (Foreign Key referencing User.UserID)
- ExerciseType (e.g., running, cycling, swimming)
- Date
- Duration
- CaloriesBurned
- DistanceCovered

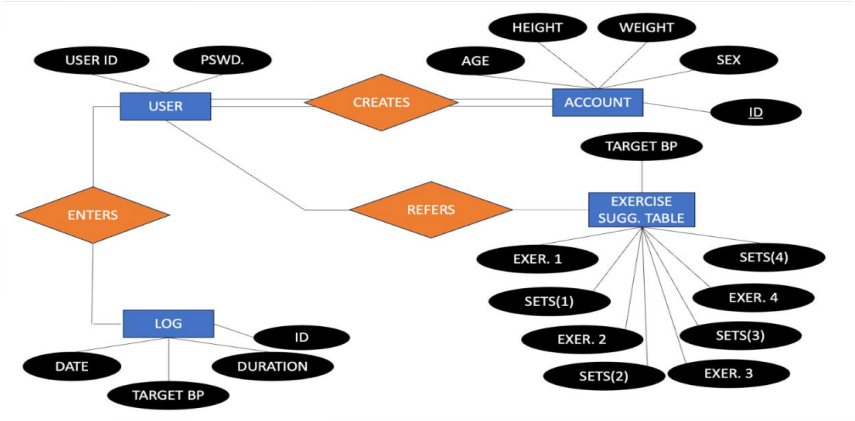
**Result:** Gathering project requirements is completed.

| Exp: 3 | Design Using ER Diagram |
|---|---|

**Aim:** To work on project designing using ER diagram.

## ER diagram:

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how "entities" such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases.

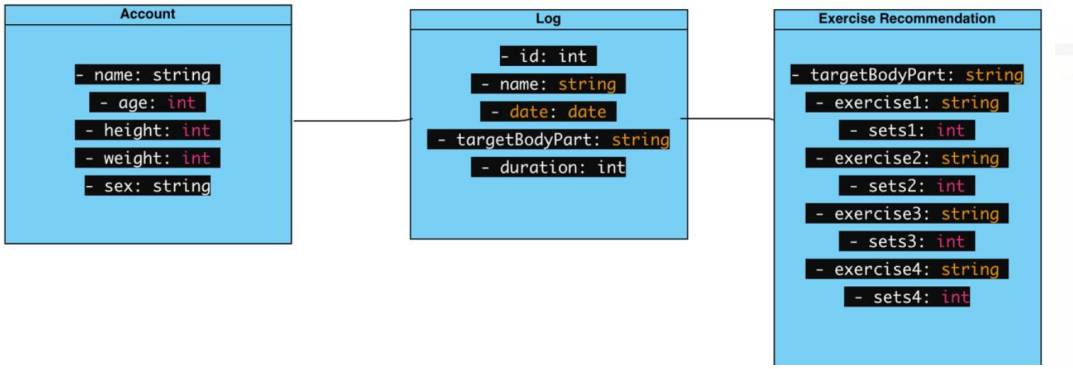



**Result:** Therefore, an ER diagram for the topic Fitness Tracker System Database System has been designed successfully based on the given requirements.

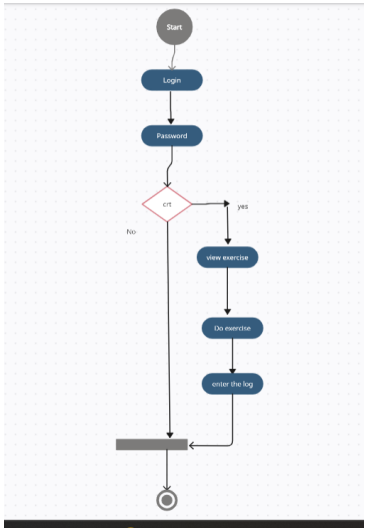
| Exp: 4 | Design Using UML Diagram |
|---|---|

**Aim:** To work on project designing using ULM diagrams.

## 1) Class Diagram:

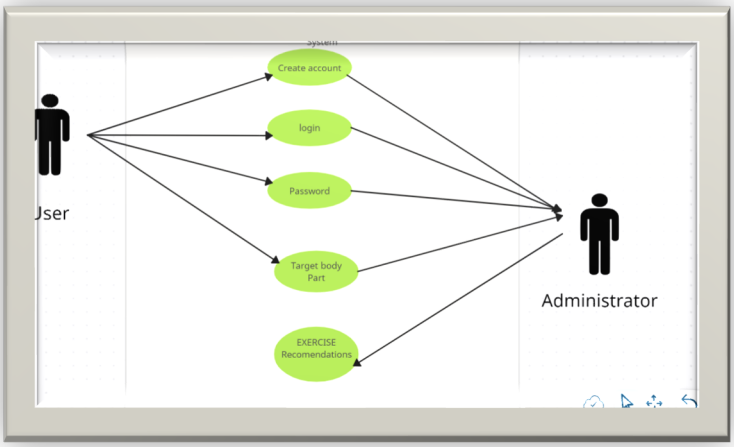


## 2) Activity Diagram




## 3) Sequence Diagram

## 4) Use-Case Diagram




**RESULT:**

To work on project designing using UML diagrams is successfully completed.

| Exp: 5 | Working with DDL and DML Commands |
|---|---|

**Aim:** To work with DDL & DML commands.

1. **DDL COMMANDS:-**

a. **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).

```
CREATE TABLE account (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    age INT NOT NULL,
    weight FLOAT NOT NULL,
    height FLOAT NOT NULL,
    sex ENUM('Male', 'Female') NOT NULL
);
```

b. **DROP:** This command is used to delete objects from the database.

```
CREATE TABLE account1 (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    age INT NOT NULL,
    weight FLOAT NOT NULL,
    height FLOAT NOT NULL,
    sex ENUM('Male', 'Female') NOT NULL
);
drop table account1;
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 09:03:23 | CREATE TABLE account1 ( id INT AUTO_INCREMENT PRIMARY KEY, name VARC... | 0 row(s) affected |
| 2 | 09:03:36 | drop table account1 | 0 row(s) affected |

c. **ALTER:** This is used to alter the structure of the database.

```
drop table account1;
Alter table account1 ADD email varchar(20);
```

Action Output

| # | Time | Action | Message | Duratic |
|---|---|---|---|---|
| 1 | 09:03:23 | CREATE TABLE account1 ( id INT AUTO_INCREMENT... | 0 row(s) affected | 0.031 s |
| 2 | 09:03:36 | drop table account1 | 0 row(s) affected | 0.016 s |
| 3 | 09:07:37 | Alter table account1 ADD email varchar(20) | Error Code: 1146. Table 'dbms_prj.account1' doesn't exist | 0.015 s |
| 4 | 09:08:10 | CREATE TABLE account1 ( id INT AUTO_INCREMENT... | 0 row(s) affected | 0.015 s |
| 5 | 09:08:13 | Alter table account1 ADD email varchar(20) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.031 s |

d. **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.

```
Truncate table account1;
```

Action Output

| # | Time | Action | Message | Dura |
|---|---|---|---|---|
| 2 | 09:03:36 | drop table account1 | 0 row(s) affected | 0.01 |
| 3 | 09:07:37 | Alter table account1 ADD email varchar(20) | Error Code: 1146. Table 'dbms_prj.account1' doesn't exist | 0.01 |
| 4 | 09:08:10 | CREATE TABLE account1 ( id INT AUTO_INCREME... | 0 row(s) affected | 0.01 |
| 5 | 09:08:13 | Alter table account1 ADD email varchar(20) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.03 |
| 6 | 09:09:32 | Truncate table account1 | 0 row(s) affected | 0.03 |

## 2)DML COMMANDS:-

a. **INSERT**: It is used to insert data into a table.

```
INSERT INTO account (name, age, weight, height, sex) VALUES
('John Smith', 30, 75.5, 180.3, 'Male'),
('Emily Johnson', 25, 62.1, 165.7, 'Female'),
('Michael Brown', 40, 80.0, 175.0, 'Male'),
('Jessica Davis', 35, 55.8, 160.0, 'Female'),
('William Wilson', 28, 68.9, 172.5, 'Male'),
```

b. **UPDATE:** It is used to update existing data within a table.

```
104 •  update account set weight=50 where id=10;
105
```

Output

Action Output

| # | Time | Action | Message | Durati |
|---|---|---|---|---|
| 4 | 09:08:10 | CREATE TABLE account1 (  id INT AUTO_INCREME... | 0 row(s) affected | 0.015 |
| 5 | 09:08:13 | Alter table account1 ADD email varchar(20) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.031 |
| 6 | 09:09:32 | Truncate table account1 | 0 row(s) affected | 0.031 |
| 7 | 09:10:53 | delete from account1 where id=1 | 0 row(s) affected | 0.015 |
| 8 | 09:11:52 | update account set weight=50 where id=10 | 1 row(s) affected Rows matched: 1 Changed: 1 Warning... | 0.016 |

**DELETE:** It is used to delete records from a database table.

```
3 •  delete from account1 where id=1;
4
```

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 3 | 09:07:37 | Alter table account1 ADD email varchar(20) | Error Code: 1146. Table 'dbms_prj.account1' doesn't exist |
| 4 | 09:08:10 | CREATE TABLE account1 (  id INT AUTO_INCREME... | 0 row(s) affected |
| 5 | 09:08:13 | Alter table account1 ADD email varchar(20) | 0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0 |
| 6 | 09:09:32 | Truncate table account1 | 0 row(s) affected |
| 7 | 09:10:53 | delete from account1 where id=1 | 0 row(s) affected |

**Result:** Thus, working with DDL & DML commands on MySQL is successfully verified.

| Exp: 6 | Working With Join, Set Operations and Aggregate Functions |
|---|---|

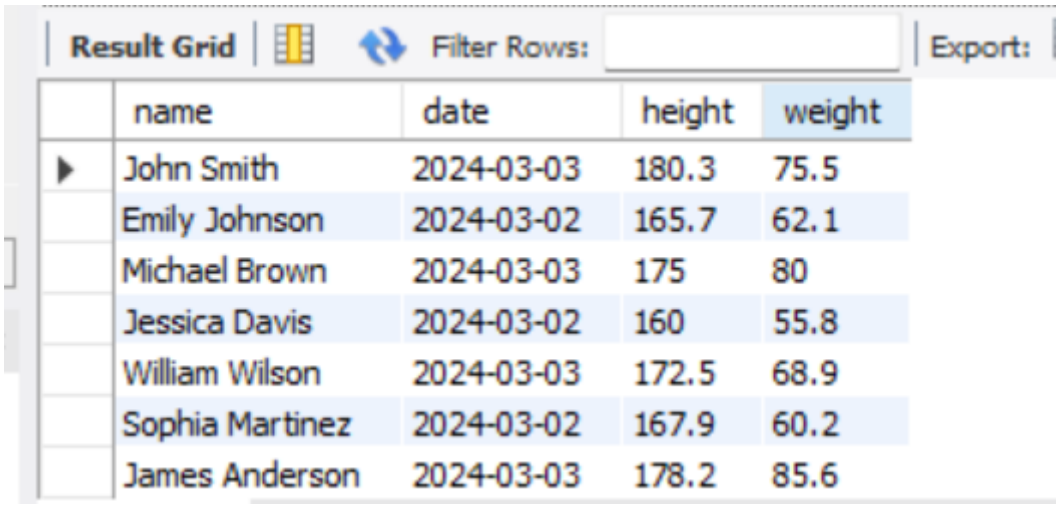
**Aim:** To perform join, set and aggregate functions on MySQL.

Retrieve the name and date of all logs along with the corresponding height and weight from the account table for each person who logged their target_body_part as 'legs'.

1)sql

```
SELECT l.name, l.date, a.height, a.weight
FROM log l
JOIN account a ON l.name = a.name
WHERE l.target_body_part = 'legs';
```

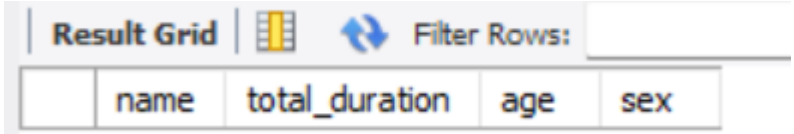
Find the total duration spent by each person on logs where the target_body_part is 'arms', and include the person's age and sex from the account table.


Result Grid | Filter Rows: | Export:

| name | date | height | weight |
|---|---|---|---|
| John Smith | 2024-03-03 | 180.3 | 75.5 |
| Emily Johnson | 2024-03-02 | 165.7 | 62.1 |
| Michael Brown | 2024-03-03 | 175 | 80 |
| Jessica Davis | 2024-03-02 | 160 | 55.8 |
| William Wilson | 2024-03-03 | 172.5 | 68.9 |
| Sophia Martinez | 2024-03-02 | 167.9 | 60.2 |
| James Anderson | 2024-03-03 | 178.2 | 85.6 |

2)

```
SELECT l.name, SUM(l.duration) AS total_duration, a.age, a.sex
FROM log l
JOIN account a ON l.name = a.name
WHERE l.target_body_part = 'arms'
GROUP BY l.name, a.age, a.sex;
```


Result Grid | Filter Rows:

| name | total_duration | age | sex |
|---|---|---|---|

3)

List all logs along with the name and height of the person who logged, but only include logs where the date is after the person's birthday.

```
SELECT l.*, a.name, a.height
FROM log l
JOIN account a ON l.name = a.name
WHERE l.date > DATE_ADD(a.age, INTERVAL -1 * YEAR(CURRENT_DATE) YEAR);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| | id | name | date | target_body_part | duration_minutes | name | height |
|---|---|---|---|---|---|---|---|

4) Retrieve the names of all people who have logged activities, but exclude those who have not logged any duration.

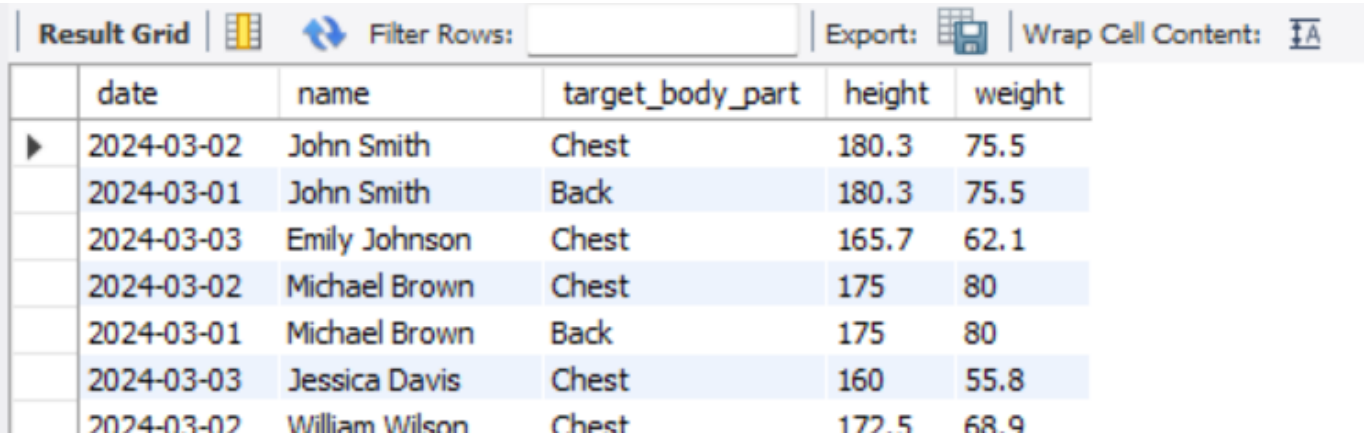
```
SELECT a.name
FROM account a
LEFT JOIN log l ON a.name = l.name
WHERE l.duration IS NOT NULL;
```

Result Grid

| | name |
|---|---|
| ▶ | John Smith |
| | John Smith |
| | John Smith |
| | John Smith |
| | Emily Johnson |
| | Emily Johnson |

5) Find the date, name, and target_body_part of all logs where the target_body_part is 'chest' or 'back', and include the height and weight of the person who logged.

```
SELECT l.date, l.name, l.target_body_part, a.height, a.weight
FROM log l
JOIN account a ON l.name = a.name
WHERE l.target_body_part IN ('chest', 'back');
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| date | name | target_body_part | height | weight |
|---|---|---|---|---|
| 2024-03-02 | John Smith | Chest | 180.3 | 75.5 |
| 2024-03-01 | John Smith | Back | 180.3 | 75.5 |
| 2024-03-03 | Emily Johnson | Chest | 165.7 | 62.1 |
| 2024-03-02 | Michael Brown | Chest | 175 | 80 |
| 2024-03-01 | Michael Brown | Back | 175 | 80 |
| 2024-03-03 | Jessica Davis | Chest | 160 | 55.8 |
| 2024-03-02 | William Wilson | Chest | 172.5 | 68.9 |

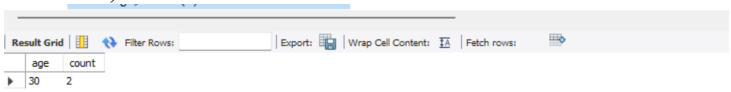
**Result:** Thus, working with Join, Set operations and Aggregate functions on MySQL is successfully verified.

| Exp: 7 | Working With Queries Related to Project |
|---|---|

**Aim:** To work with queries related to project on MySQL.

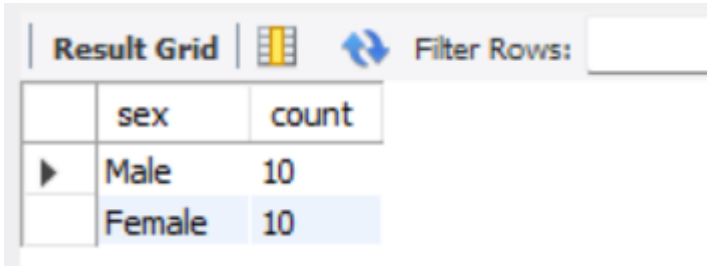
1) #1 Most common age among the individuals:

```
SELECT age, COUNT(*) AS count FROM ACCOUNT
GROUP BY age
ORDER BY count DESC
LIMIT 1;
```



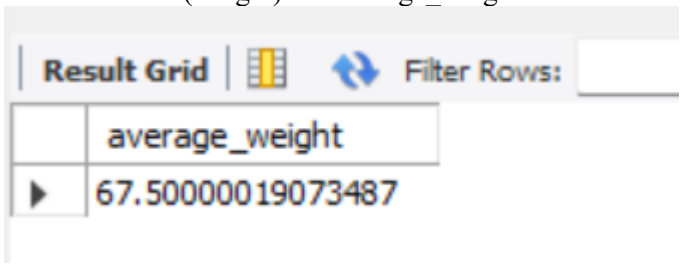

#2 Number of males and females:

```
SELECT sex, COUNT(*) AS count FROM ACCOUNT
GROUP BY sex;
```



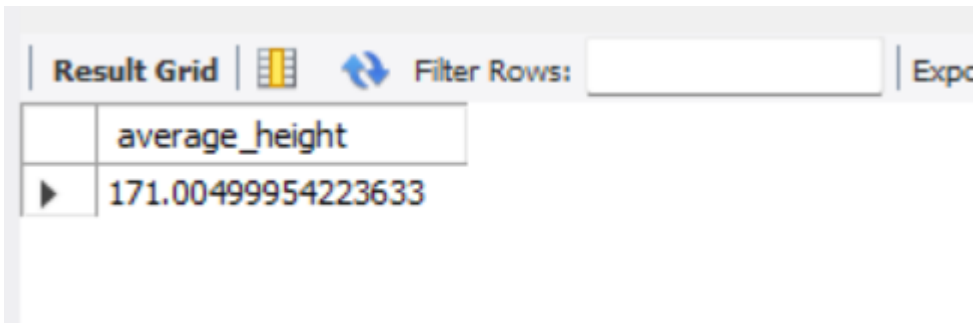

#3 Average weight of the individuals:

```
SELECT AVG(weight) AS average_weight FROM ACCOUNT;
```




#4
Average height of the individuals:

```
SELECT AVG(height) AS average_height FROM ACCOUNT;
```




#6 Oldest individual in the table:

```
SELECT * FROM ACCOUNT
ORDER BY age DESC
LIMIT 1;
```

Result Grid | Filter Rows: | Edit:

| id | name | age | weight | height | sex |
|---|---|---|---|---|---|
| 7 | James Anderson | 45 | 85.6 | 178.2 | Male |

#7 Youngest individual in the table:

```
SELECT * FROM ACCOUNT
ORDER BY age ASC
LIMIT 1;
```

Result Grid | Filter Rows: | Edit:

| id | name | age | weight | height | sex |
|---|---|---|---|---|---|
| 20 | Amelia Evans | 23 | 54.7 | 161.3 | Female |

#8 Total weight of all individuals:

```
SELECT SUM(weight) AS total_weight FROM ACCOUNT;
```

Result Grid | Filter Rows:

| total_weight |
|---|
| 1350.0000038146973 |

#9 Total height of all individuals:

```
SELECT SUM(height) AS total_height FROM ACCOUNT;
```

Result Grid | Filter Rows:

| total_height |
|---|
| 3420.0999908447266 |

#10 Number of individuals within a certain age range (e.g., 30 to 40):

```
SELECT COUNT(*) AS count_in_range FROM ACCOUNT
WHERE age BETWEEN 30 AND 40;
```

Result Grid | Filter Rov

| count_in_range |
|---|
| 11 |

#11 Average weight of males and females separately:

```
SELECT sex, AVG(weight) AS average_weight FROM ACCOUNT
GROUP BY sex;
```

Result Grid | Filter Rows:

| sex | average_weight |
|---|---|
| Male | 75.98000030517578 |
| Female | 59.02000007629395 |

#12 Average height of males and females separately:

SELECT sex, AVG(height) AS average_height FROM ACCOUNT
GROUP BY sex;

Result Grid | Filter Rows:

| sex | average_height |
|---|---|
| Male | 177.08999938964843 |
| Female | 164.9199996948242 |

#13 Total number of records in the table:

SELECT COUNT(*) AS total_records FROM ACCOUNT;

Result Grid | Filter Rows

| total_records |
|---|
| 20 |

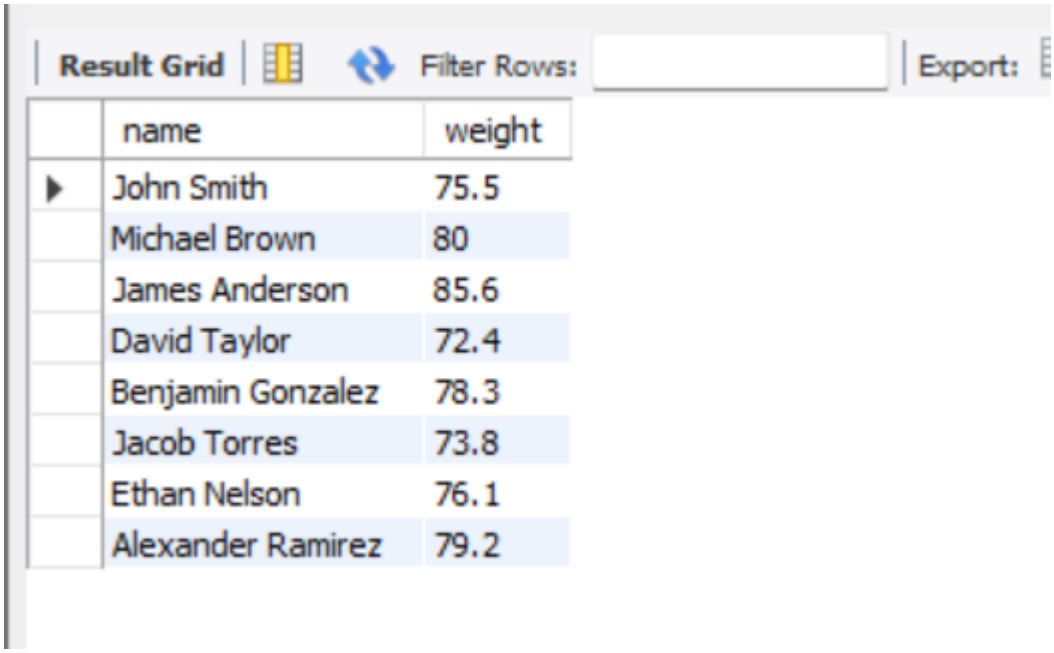
#14 BMI (Body Mass Index) of each individual:
SELECT id, name, weight, height, (weight / (height * height)) AS bmi FROM ACCOUNT;

Result Grid | Filter Rows: | Export: | Wrap Cell C

| id | name | weight | height | bmi |
|---|---|---|---|---|
| 1 | John Smith | 75.5 | 180.3 | 0.0023224987209083665 |
| 2 | Emily Johnson | 62.1 | 165.7 | 0.0022617603619966777 |
| 3 | Michael Brown | 80 | 175 | 0.0026122448979591837 |
| 4 | Jessica Davis | 55.8 | 160 | 0.0021796874701976777 |
| 5 | William Wilson | 68.9 | 172.5 | 0.0023154799924681004 |
| 6 | Sophia Martinez | 60.2 | 167.9 | 0.0021354781693395442 |
| 7 | James Anderson | 85.6 | 178.2 | 0.0026956181532874791 |
| 8 | Olivia Garcia | 57.3 | 163.4 | 0.0021461029139708434 |
| 9 | David Taylor | 72.4 | 177.8 | 0.0022902086316994247 |
| 10 | Emma Rodriguez | 63.7 | 169.1 | 0.0022276768464119 47 |

0.00213

#15 displaying male above the weight of 70
select name,weight from account where sex="male" and weight>70;
select * from account;

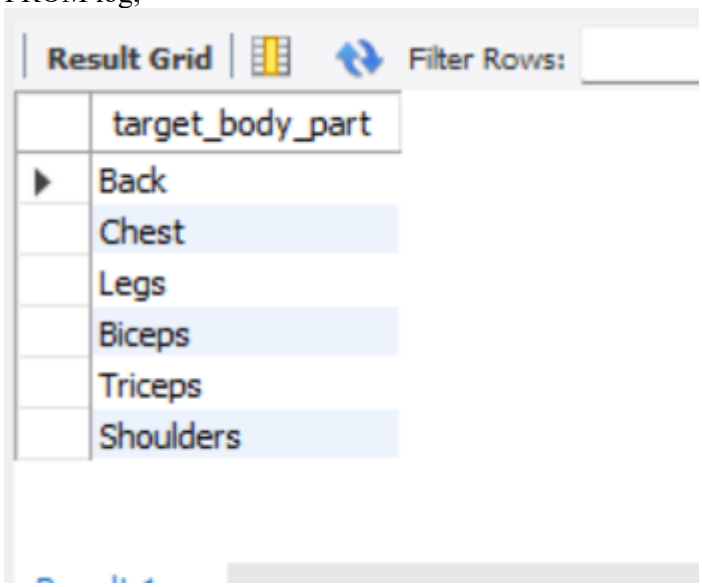


Result Grid | Filter Rows: | Export:

| name | weight |
|---|---|
| John Smith | 75.5 |
| Michael Brown | 80 |
| James Anderson | 85.6 |
| David Taylor | 72.4 |
| Benjamin Gonzalez | 78.3 |
| Jacob Torres | 73.8 |
| Ethan Nelson | 76.1 |
| Alexander Ramirez | 79.2 |

#16 List all distinct exercises in the table.

```
SELECT distinct(target_body_part)
FROM log;
```

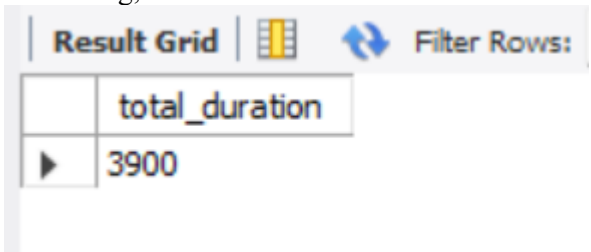


Result Grid | Filter Rows:

| target_body_part |
|---|
| Back |
| Chest |
| Legs |
| Biceps |
| Triceps |
| Shoulders |

#17 Find the total duration of all workouts in the table.

```
SELECT SUM(duration_minutes) AS total_duration
FROM log;
```

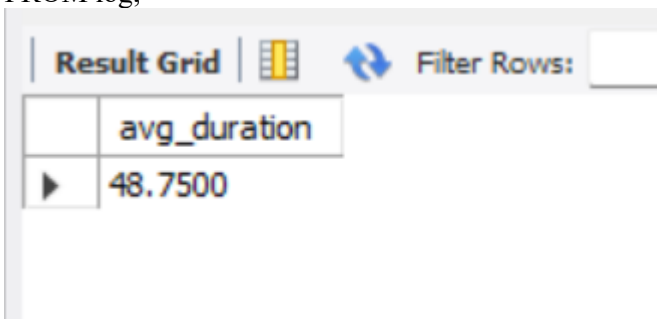


Result Grid | Filter Rows:

| total_duration |
|---|
| 3900 |

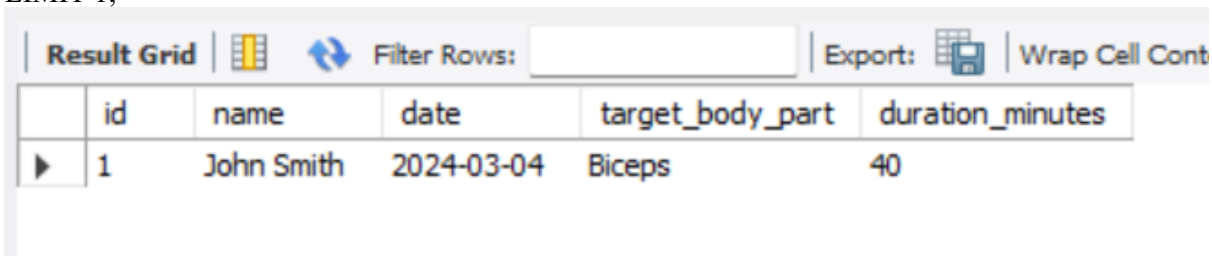
#18 What was the average duration of workouts in minutes?

SELECT AVG(duration_minutes) AS avg_duration
FROM log;


Result Grid | Filter Rows:

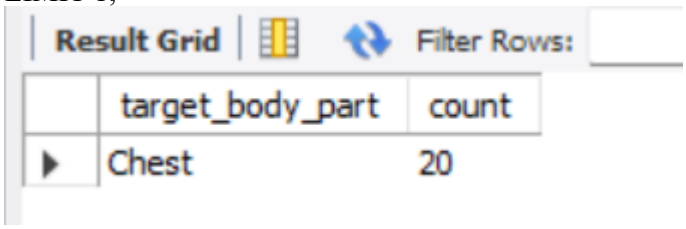
| avg_duration |
| --- |
| 48.7500 |

#19 Find the shortest workout in the table.
SELECT *
FROM log
ORDER BY duration_minutes ASC
LIMIT 1;


Result Grid | Filter Rows: | Export: | Wrap Cell Cont

| id | name | date | target_body_part | duration_minutes |
| --- | --- | --- | --- | --- |
| 1 | John Smith | 2024-03-04 | Biceps | 40 |

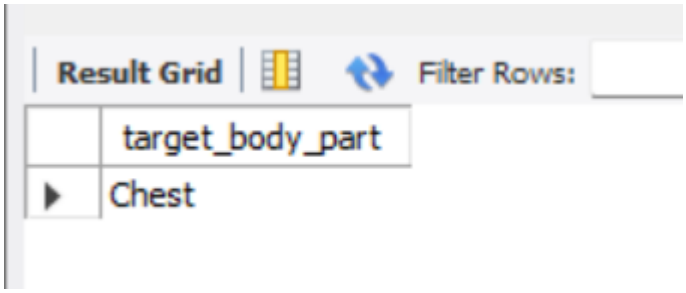
#20 What was the most common target body part exercised?
SELECT target_body_part, COUNT(*) AS count
FROM log
GROUP BY target_body_part
ORDER BY count DESC
LIMIT 1;


Result Grid | Filter Rows:

| target_body_part | count |
| --- | --- |
| Chest | 20 |

#21 Find all the exercises performed by Jessica Davis on March 3rd, 2024.

SELECT target_body_part
FROM log
WHERE name = 'Jessica Davis'
AND date = '2024-03-03';


Result Grid | Filter Rows:

| target_body_part |
| --- |
| Chest |

#22 Which person exercised the most legs in total minutes?

```
SELECT name, SUM(CASE WHEN target_body_part = 'Legs' THEN duration_minutes ELSE 0 END) AS
total_leg_minutes
FROM log
GROUP BY name
ORDER BY total_leg_minutes DESC
LIMIT 1;
```

Result Grid | Filter Rows:

| name | total_leg_minutes |
|---|---|
| John Smith | 60 |

#23 Find all the workouts that were longer than 45 minutes.

```
SELECT *
FROM log
WHERE duration_minutes > 45;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

| id | name | date | target_body_part | duration_minutes |
|---|---|---|---|---|
| 1 | John Smith | 2024-03-01 | Back | 50 |
| 1 | John Smith | 2024-03-03 | Legs | 60 |
| 2 | Emily Johnson | 2024-03-02 | Legs | 60 |
| 2 | Emily Johnson | 2024-03-04 | Shoulders | 50 |
| 3 | Michael Brown | 2024-03-01 | Back | 50 |
| 3 | Michael Brown | 2024-03-03 | Legs | 60 |
| 4 | Jessica Davis | 2024-03-02 | Legs | 60 |
| 4 | Jessica Davis | 2024-03-04 | Shoulders | 50 |

#24 Find all the exercises performed between March 2nd and March 4th, 2024.

```
SELECT *
FROM log
WHERE date BETWEEN '2024-03-02' AND '2024-03-04';
select * from log;
```

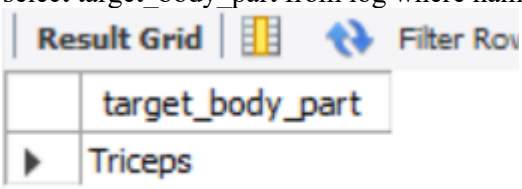
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| id | name | date | target_body_part | duration_minutes |
|---|---|---|---|---|
| 1 | John Smith | 2024-03-02 | Chest | 45 |
| 1 | John Smith | 2024-03-03 | Legs | 60 |
| 1 | John Smith | 2024-03-04 | Biceps | 40 |
| 2 | Emily Johnson | 2024-03-02 | Legs | 60 |
| 2 | Emily Johnson | 2024-03-03 | Chest | 45 |
| 2 | Emily Johnson | 2024-03-04 | Shoulders | 50 |
| 3 | Michael Brown | 2024-03-02 | Chest | 45 |
| 3 | Michael Brown | 2024-03-03 | Legs | 60 |

log 9 ×

#25 Find the exercise performed by isabella on 2024-03-01;

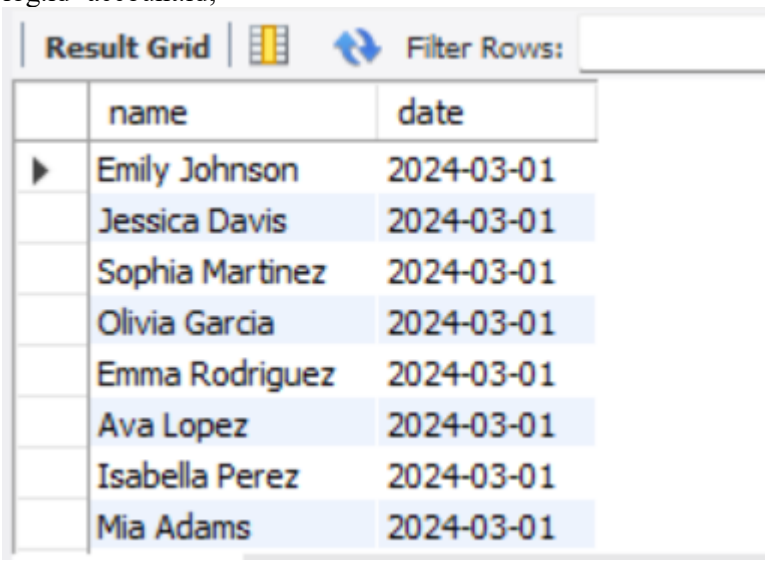
select target_body_part from log where name="Isabella Perez" and date="2024-03-01";



| target_body_part |
| --- |
| Triceps |

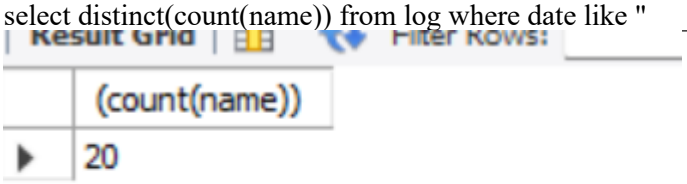
#26 display all the female's doing bicep exercise

select account.name,log.date from account,log where sex="female" and target_body_part="triceps" and log.id=account.id;



| name | date |
| --- | --- |
| Emily Johnson | 2024-03-01 |
| Jessica Davis | 2024-03-01 |
| Sophia Martinez | 2024-03-01 |
| Olivia Garcia | 2024-03-01 |
| Emma Rodriguez | 2024-03-01 |
| Ava Lopez | 2024-03-01 |
| Isabella Perez | 2024-03-01 |
| Mia Adams | 2024-03-01 |

#27 how many people visited on march 1st
select distinct(count(name)) from log where date like "  __-__-01";



| (count(name)) |
| --- |
| 20 |

#28 how long did jhon smith spend at the gym

select sum(duration_minutes) from log where name="John Smith";

select * from log;

Result Grid | Filter Rows:

| sum(duration_minutes) |
|---|
| 195 |

#29 Total time spent by female at the gym

select sum(duration_minutes) from log,account where sex="female" and log.id=account.id;

Result Grid | Filter Rows:

| sum(duration_minutes) |
|---|
| 1950 |

#30 displaying the count of male who trained their back;
select count(a.name) from log a,account b where target_body_part="Back" and sex="male" and a.id=b.id;

Result Grid | Filter Rows:

| count(a.name) |
|---|
| 10 |

#31. What is the SQL query to retrieve all exercises for the "Back" target body part?

SELECT * FROM exercise WHERE target_body_part = 'Back';

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

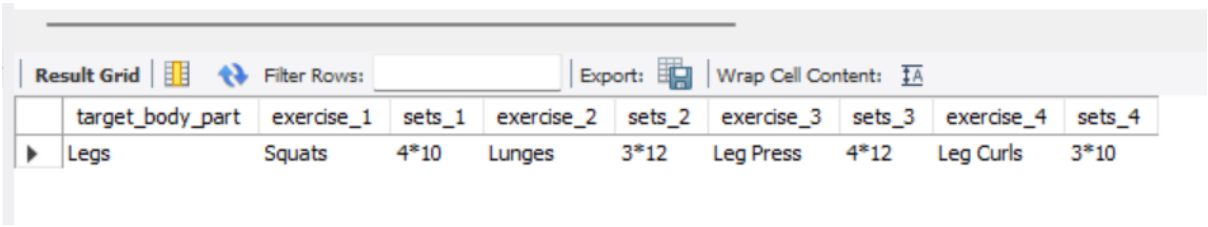
| target_body_part | exercise_1 | sets_1 | exercise_2 | sets_2 | exercise_3 | sets_3 | exercise_4 | sets_4 |
|---|---|---|---|---|---|---|---|---|
| Back | Deadlift | 3*12 | Pull-ups | 3*10 | Barbell Rows | 4*8 | Lat Pulldowns | 3*12 |

#32 What is the SQL query to retrieve all exercises for the "Shoulders" target body part?
SELECT * FROM exercise WHERE target_body_part = 'Shoulders';

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| target_body_part | exercise_1 | sets_1 | exercise_2 | sets_2 | exercise_3 | sets_3 | exercise_4 | sets_4 |
|---|---|---|---|---|---|---|---|---|
| Shoulders | Military Press | 4*8 | Lateral Raises | 3*12 | Front Raises | 4*10 | Arnold Press | 3*8 |

#33 What is the SQL query to retrieve all exercises for the "Legs" target body part?
SELECT * FROM exercise WHERE target_body_part = 'Legs';

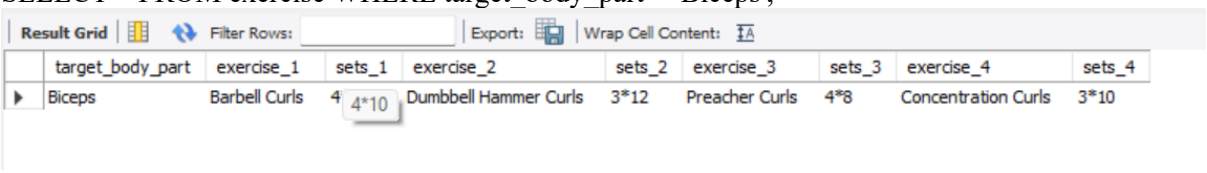


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| target_body_part | exercise_1 | sets_1 | exercise_2 | sets_2 | exercise_3 | sets_3 | exercise_4 | sets_4 |
|---|---|---|---|---|---|---|---|---|
| Legs | Squats | 4*10 | Lunges | 3*12 | Leg Press | 4*12 | Leg Curls | 3*10 |

#34 What is the SQL query to retrieve all exercises for the "Biceps" target body part?

SELECT * FROM exercise WHERE target_body_part = 'Biceps';



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

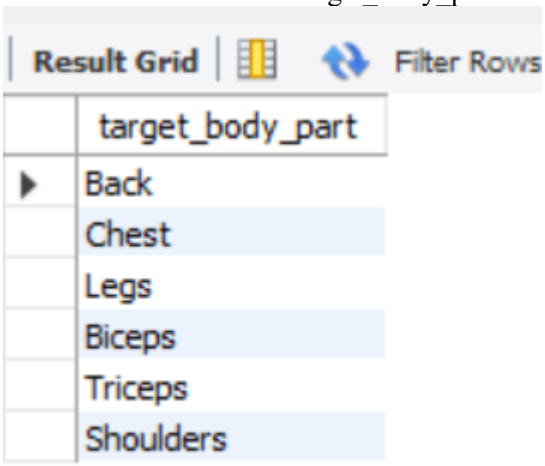
| target_body_part | exercise_1 | sets_1 | exercise_2 | sets_2 | exercise_3 | sets_3 | exercise_4 | sets_4 |
|---|---|---|---|---|---|---|---|---|
| Biceps | Barbell Curls | 4 | Dumbbell Hammer Curls | 3*12 | Preacher Curls | 4*8 | Concentration Curls | 3*10 |

4*10

select * from exercise;

#35 Write a SQL query to list all unique body parts targeted by the exercise routines.

SELECT DISTINCT target_body_part FROM exercise;

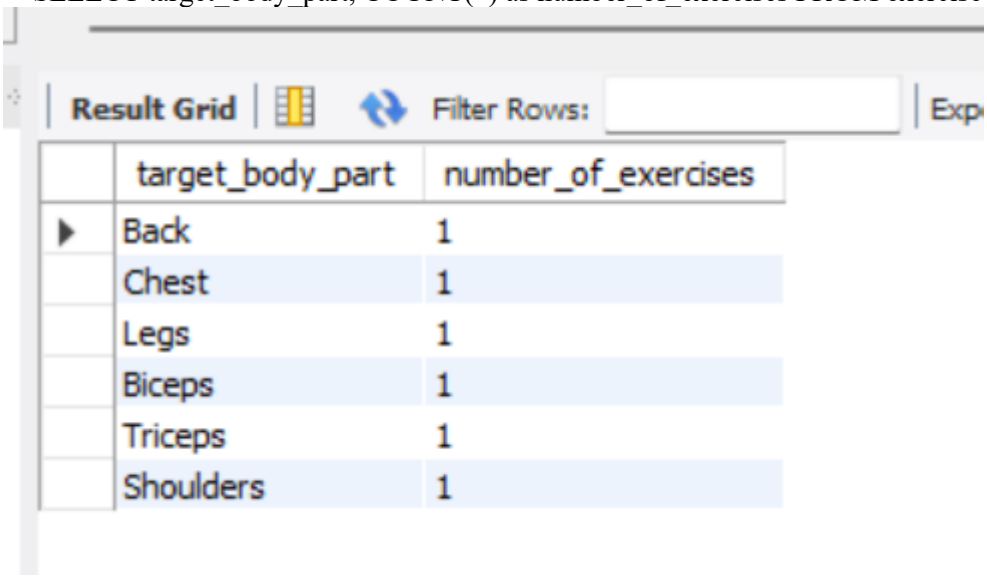


Result Grid | Filter Rows

| target_body_part |
|---|
| Back |
| Chest |
| Legs |
| Biceps |
| Triceps |
| Shoulders |

#36 Write a SQL query to count the number of exercise routines for each body part. **

SELECT target_body_part, COUNT(*) as number_of_exercises FROM exercise GROUP BY target_body_part



Result Grid | Filter Rows: | Exp

| target_body_part | number_of_exercises |
|---|---|
| Back | 1 |
| Chest | 1 |
| Legs | 1 |
| Biceps | 1 |
| Triceps | 1 |
| Shoulders | 1 |

#37 Find all exercises targeting the legs with more than 3 sets.

```
SELECT * FROM exercise
WHERE target_body_part = 'legs'
AND (sets_1 > 2 OR sets_2 > 2 OR sets_3 > 2 OR sets_4 > 2);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| target_body_part | exercise_1 | sets_1 | exercise_2 | sets_2 | exercise_3 | sets_3 | exercise_4 | sets_4 |
|---|---|---|---|---|---|---|---|---|
| Legs | Squats | 4*10 | Lunges | 3*12 | Leg Press | 4*12 | Leg Curls | 3*10 |

#38 List exercises targeting the back ordered by the number of sets.

```
SELECT * FROM exercise
WHERE target_body_part = 'back'
ORDER BY sets_1 + sets_2 + sets_3 + sets_4;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| target_body_part | exercise_1 | sets_1 | exercise_2 | sets_2 | exercise_3 | sets_3 | exercise_4 | sets_4 |
|---|---|---|---|---|---|---|---|---|
| Back | Deadlift | 3*12 | Pull-ups | 3*10 | Barbell Rows | 4*8 | Lat Pulldowns | 3*12 |

#39 What are the exercises targeting the biceps or triceps,
# and what are their respective sets, ordered by the number of sets in descending order?

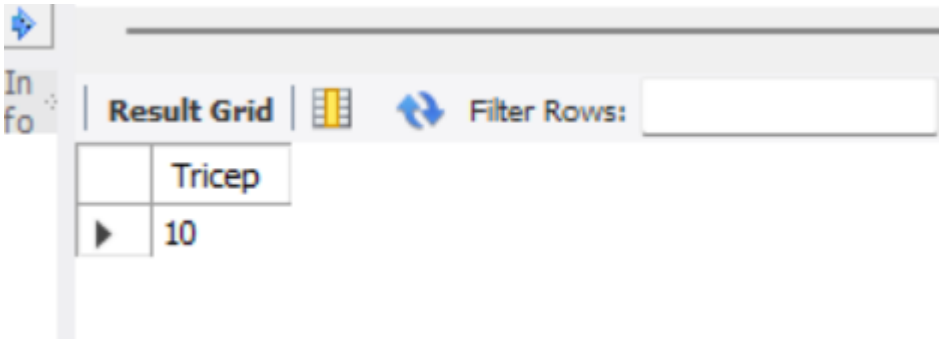
```
SELECT target_body_part,
      exercise_1, sets_1,
      exercise_2, sets_2,
      exercise_3, sets_3,
      exercise_4, sets_4
FROM exercise
WHERE target_body_part IN ('Biceps', 'Triceps')
ORDER BY sets_1 + sets_2 + sets_3 + sets_4 DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| target_body_part | exercise_1 | sets_1 | exercise_2 | sets_2 | exercise_3 | sets_3 | exercise_4 | sets_4 |
|---|---|---|---|---|---|---|---|---|
| Biceps | Barbell Curls | 4*10 | Dumbbell Hammer Curls | 3*12 | Preacher Curls | 4*8 | Concentration Curls | 3*10 |
| Triceps | Tricep Dips | 3*10 | Skull Crushers | 4*8 | Tricep Pushdowns | 3*12 | Overhead Tricep Extensions | 4*10 |

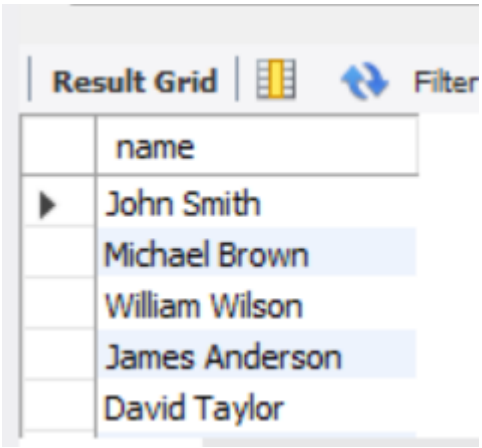
40) displaying the customers who exercised their triceps

```
select count(a.target_body_part) as Tricep from log a,account b where target_body_part="Triceps" and a.id=b.id;
```




41) people who have done legs on 04/03/2024

```
select name from log where target_body_part="Biceps" and date="2024-03-04";
```




**Result:** Thus, working with queries related to project on MySQL is successfully verified.

| Exp: 8 | Working With Basic PL/SQL Programming |
|---|---|

**Aim:** To work with basic PL/SQL programming on MySQL.

1. Write a PL/SL program to check given number is prime or not.

```
mysql> DELIMITER //
mysql> CREATE FUNCTION IsPrime(N INT) RETURNS VARCHAR(50) READS SQL DATA
    -> BEGIN
    ->     DECLARE I INT;
    ->     DECLARE TEMP INT;
    ->     SET I = 2;
    ->     SET TEMP = 0;
    ->
    ->     WHILE I <= N/2 DO
    ->         IF N % I = 0 THEN
    ->             SET TEMP = 1;
    ->             RETURN 'IT IS NOT A PRIME NUMBER';
    ->         END IF;
    ->         SET I = I + 1;
    ->     END WHILE;
    ->
    ->     IF TEMP = 1 THEN
    ->         RETURN 'IT IS NOT A PRIME NUMBER';
    ->     ELSE
    ->         RETURN 'IT IS A PRIME NUMBER';
    ->     END IF;
    ->
    -> END//
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT IsPrime(7);
+---------------------+
| IsPrime(7)          |
+---------------------+
| IT IS A PRIME NUMBER |
+---------------------+
1 row in set (0.01 sec)
```

2. To Find Factorial

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION CalculateFactorial(n INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE result INT DEFAULT 1;
    ->   DECLARE i INT DEFAULT 1;
    ->
    ->   WHILE i <= n DO
    ->     SET result = result * i;
    ->     SET i = i + 1;
    ->   END WHILE;
    ->
    ->   RETURN result;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT CalculateFactorial(5);
+----------------------+
| CalculateFactorial(5) |
+----------------------+
|                  120 |
+----------------------+
1 row in set (0.00 sec)
```

3. Write a PL/SL program to reverser a given number

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION ReverseNumber(n INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE reversed INT DEFAULT 0;
    ->   DECLARE digit INT;
    ->
    ->   WHILE n > 0 DO
    ->     SET digit = n % 10;
    ->     SET reversed = reversed * 10 + digit;
    ->     SET n = FLOOR(n / 10);
    ->   END WHILE;
    ->
    ->   RETURN reversed;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
```

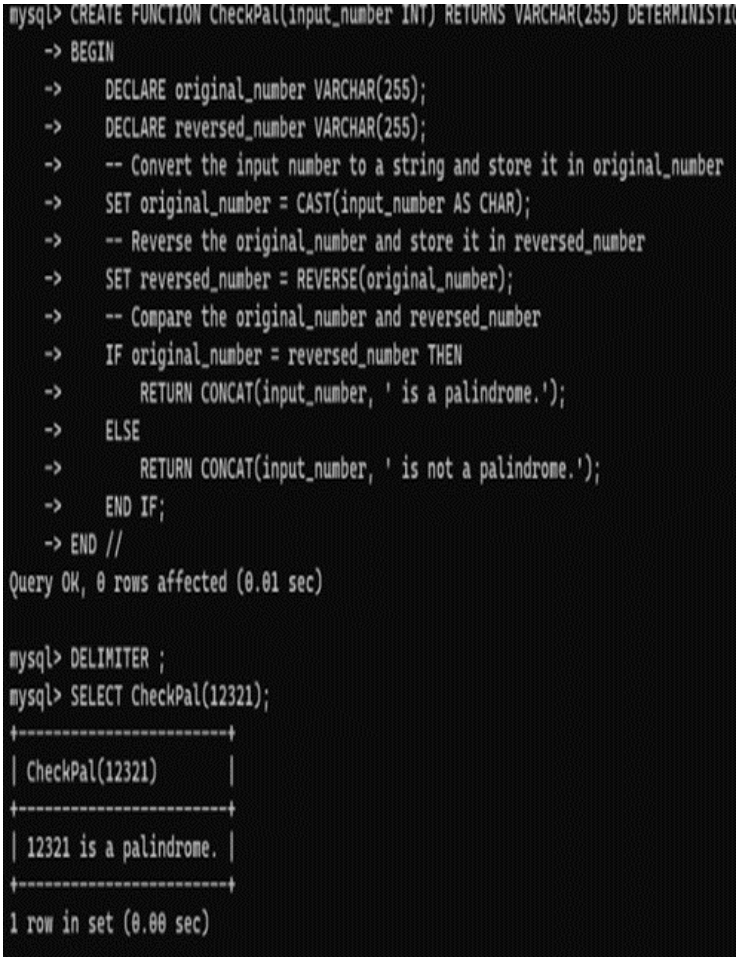
```
mysql> Select ReverseNumber(9786);
+----------------------+
| ReverseNumber(9786) |
+----------------------+
|                 6879 |
+----------------------+
1 row in set (0.00 sec)
```

4. Write a PL/SL program to generate a FibonacPOci series

```
mysql> CREATE FUNCTION FibonacciSeries(terms INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->   DECLARE result VARCHAR(255) DEFAULT '0, 1';
    ->   DECLARE i INT DEFAULT 2;
    ->   DECLARE a INT DEFAULT 0;
    ->   DECLARE b INT DEFAULT 1;
    ->   DECLARE next_term INT;
    ->
    ->   IF terms <= 0 THEN
    ->     RETURN 'Invalid input';
    ->   ELSEIF terms = 1 THEN
    ->     RETURN '0';
    ->   ELSEIF terms = 2 THEN
    ->     RETURN '0, 1';
    ->   END IF;
    ->
    ->   WHILE i < terms DO
    ->     SET next_term = a + b;
    ->     SET result = CONCAT(result, ', ', next_term);
    ->     SET a = b;
    ->     SET b = next_term;
    ->     SET i = i + 1;
    ->   END WHILE;
    ->
    ->   RETURN result;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT FibonacciSeries(10);
+------------------------------+
| FibonacciSeries(10)          |
+------------------------------+
| 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 |
+------------------------------+
1 row in set (0.00 sec)
```

5. Write a PL/SL program to check given number is palindrome or not



```
mysql> CREATE FUNCTION CheckPal(input_number INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE original_number VARCHAR(255);
    ->     DECLARE reversed_number VARCHAR(255);
    ->     -- Convert the input number to a string and store it in original_number
    ->     SET original_number = CAST(input_number AS CHAR);
    ->     -- Reverse the original_number and store it in reversed_number
    ->     SET reversed_number = REVERSE(original_number);
    ->     -- Compare the original_number and reversed_number
    ->     IF original_number = reversed_number THEN
    ->         RETURN CONCAT(input_number, ' is a palindrome.');
    ->     ELSE
    ->         RETURN CONCAT(input_number, ' is not a palindrome.');
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT CheckPal(12321);
+----------------------+
| CheckPal(12321)      |
+----------------------+
| 12321 is a palindrome. |
+----------------------+
1 row in set (0.00 sec)
```

6. CalculateSal

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION CalculateSavings(salary INT, shifts INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE savings INT;
    ->   DECLARE food_expense INT;
    ->   DECLARE travel_expense INT;
    ->   DECLARE shift_earnings INT;
    ->
    ->   -- Check if the salary is too large
    ->   IF salary > 8000 THEN
    ->     RETURN -1; -- Salary too large
    ->   END IF;
    ->
    ->   -- Check if the shifts are too small
    ->   IF shifts < 0 THEN
    ->     RETURN -2; -- Shifts too small
    ->   END IF;
    ->
    ->   -- Check if the salary is too small
    ->   IF salary < 0 THEN
    ->     RETURN -3; -- Salary too small
    ->   END IF;
    ->
    ->   -- Calculate the amount spent on food and travel
    ->   SET food_expense = salary * 20 / 100;
    ->   SET travel_expense = salary * 30 / 100;
    ->
    ->   -- Calculate the amount earned from shifts
    ->   SET shift_earnings = shifts * salary * 2 / 100;
    ->
    ->   -- Calculate savings
    ->   SET savings = salary - food_expense - travel_expense + shift_earnings;
    ->
    ->   RETURN savings;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT CalculateSavings(5000, 5);
+--------------------------+
| CalculateSavings(5000, 5) |
+--------------------------+
|                     3000 |
+--------------------------+
1 row in set (0.00 sec)
```

7. Product of Digits

```
mysql> DELIMITER //
mysql> CREATE FUNCTION productDigits(input_num INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE product INT;
    ->     DECLARE digit INT;
    ->     -- Check for invalid input conditions
    ->     IF input_num < 0 OR input_num > 32767 THEN
    ->         RETURN 'Invalid Input'; -- Return the message as a string
    ->     END IF;
    ->     SET product = 1;
    ->     -- Calculate the product of digits
    ->     WHILE input_num > 0 DO
    ->         SET digit = input_num % 10;
    ->         SET product = product * digit;
    ->         SET input_num = FLOOR(input_num / 10);
    ->     END WHILE;
    ->     RETURN CAST(product AS CHAR);
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> SELECT productDigits(45);
+------------------+
| productDigits(45) |
+------------------+
| 20               |
+------------------+
1 row in set (0.00 sec)
```

8. Power of Two

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION IsPowerOfTwo(num INT) RETURNS VARCHAR(3) DETERMINISTIC
    -> BEGIN
    ->   IF num < 0 THEN
    ->     RETURN 'Number too small';
    ->   END IF;
    ->
    ->   IF num > 32767 THEN
    ->     RETURN 'Number too large';
    ->   END IF;
    ->
    ->   IF num > 0 AND (num & (num - 1)) = 0 THEN
    ->     RETURN 'Yes'; -- It's a power of 2
    ->   ELSE
    ->     RETURN 'No'; -- It's not a power of 2
    ->   END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT IsPowerOfTwo(18);
+-----------------+
| IsPowerOfTwo(18) |
+-----------------+
| No              |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT IsPowerOfTwo(8);
+----------------+
| IsPowerOfTwo(8) |
+----------------+
| Yes            |
+----------------+
1 row in set (0.00 sec)
```

9. Decimal conversion

```
mysql> DELIMITER //
mysql> CREATE FUNCTION convertToDecimal(bin_num VARCHAR(10)) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE decimal_num INT DEFAULT 0;
    ->     DECLARE binary_digit CHAR(1);
    ->     DECLARE i INT;
    ->     -- Check for invalid input conditions
    ->     IF bin_num REGEXP '^[01]+$' = 0 OR LENGTH(bin_num) > 5 THEN
    ->         RETURN 'Invalid Input'; -- Return the message as a string
    ->     END IF;
    ->     -- Convert binary to decimal
    ->     SET i = LENGTH(bin_num);
    ->     WHILE i > 0 DO
    ->         SET binary_digit = SUBSTRING(bin_num, i, 1);
    ->         SET decimal_num = decimal_num + (binary_digit * POW(2, LENGTH(bin_num) - i));
    ->         SET i = i - 1;
    ->     END WHILE;
    ->     RETURN CAST(decimal_num AS CHAR);
    -> END //
Query OK, 0 rows affected (0.06 sec)

mysql> DELIMITER ;
mysql> SELECT  convertToDecimal(1001);
+----------------------+
| convertToDecimal(1001) |
+----------------------+
| 9                    |
+----------------------+
1 row in set (0.03 sec)
```

10. Arithmetic Operation

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION PerformArithmeticOperation(input1 INT, input2 INT, operation INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   -- Check if input1 or input2 is negative or greater than 32767
    ->   IF input1 < 0 OR input1 > 32767 OR input2 < 0 OR input2 > 32767 THEN
    ->     RETURN -1;
    ->   END IF;
    ->
    ->   -- Check if the operation choice is in the range 1 to 4
    ->   IF operation < 1 OR operation > 4 THEN
    ->     RETURN -1;
    ->   END IF;
    ->
    ->   -- Perform the specified arithmetic operation
    ->   CASE operation
    ->     WHEN 1 THEN RETURN input1 + input2;
    ->     WHEN 2 THEN RETURN input1 - input2;
    ->     WHEN 3 THEN RETURN input1 * input2;
    ->     WHEN 4 THEN
    ->       -- Check if input2 is zero to avoid division by zero
    ->       IF input2 = 0 THEN
    ->         RETURN -1; -- Division by zero, return -1
    ->       ELSE
    ->         RETURN input1 / input2; -- Calculate the quotient
    ->       END IF;
    ->   END CASE;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT PerformArithmeticOperation(10, 5, 2);
+-------------------------------------+
| PerformArithmeticOperation(10, 5, 2) |
+-------------------------------------+
|                                   5 |
+-------------------------------------+
1 row in set (0.00 sec)
```

11. Digit Factorial

```
mysql> DELIMITER //
mysql> CREATE FUNCTION DigitFactorial(inputNumber INT) RETURNS VARCHAR(1000) DETERMINISTI
    -> BEGIN
    ->     DECLARE n INT DEFAULT inputNumber;
    ->     DECLARE factorial INT;
    ->     DECLARE result VARCHAR(1000) DEFAULT '';
    ->
    ->     WHILE n > 0 DO
    ->         SET factorial = 1;
    ->         SET @digit = n % 10;
    ->
    ->         IF @digit > 1 THEN
    ->             SET @i = 2;
    ->             WHILE @i <= @digit DO
    ->                 SET factorial = factorial * @i;
    ->                 SET @i = @i + 1;
    ->             END WHILE;
    ->         END IF;
    ->
    ->         SET result = CONCAT(result, factorial, ',');
    ->         SET n = FLOOR(n / 10);
    ->     END WHILE;
    ->
    ->     RETURN SUBSTRING(result, 1, LENGTH(result) - 1); -- Remove trailing comma
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> SELECT DigitFactorial(123);
+-------------------+
| DigitFactorial(123) |
+-------------------+
| 6,2,1             |
+-------------------+
1 row in set (0.00 sec)
```

12. Sum of Odd Digits

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION SumOddDigits(input_num INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE digit INT;
    ->   DECLARE sum_odd_digits INT DEFAULT 0;
    ->   DECLARE num INT;
    ->
    ->   -- Check if input_num is less than zero or greater than 32767
    ->   IF input_num < 0 OR input_num > 32767 THEN
    ->     RETURN -1;
    ->   END IF;
    ->
    ->   SET num = input_num;
    ->
    ->   -- Loop to extract and sum the odd digits
    ->   WHILE num > 0 DO
    ->     SET digit = num % 10;
    ->
    ->     IF digit % 2 <> 0 THEN
    ->       SET sum_odd_digits = sum_odd_digits + digit;
    ->     END IF;
    ->
    ->     SET num = FLOOR(num / 10);
    ->   END WHILE;
    ->
    ->   RETURN sum_odd_digits;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT SumOddDigits(15672);
+--------------------+
| SumOddDigits(15672) |
+--------------------+
|                 13 |
+--------------------+
1 row in set (0.00 sec)
```

13. Generate New number

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION GenerateNewNumber(input_num INT) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->   DECLARE new_num INT DEFAULT 0;
    ->   DECLARE digit INT;
    ->   DECLARE multiplier INT DEFAULT 1;
    ->
    ->   -- Check if the input_num is less than zero or greater than 32767
    ->   IF input_num < 0 OR input_num > 32767 THEN
    ->     RETURN -1;
    ->   END IF;
    ->
    ->   -- Loop through the digits of the input_num
    ->   WHILE input_num > 0 DO
    ->     SET digit = input_num % 10;
    ->
    ->     IF digit % 2 = 0 THEN
    ->       -- Even digit, replace with the next even digit
    ->       SET digit = (digit + 2) % 10;
    ->     ELSE
    ->       -- Odd digit, replace with the next odd digit
    ->       SET digit = (digit + 2) % 9;
    ->     END IF;
    ->
    ->     -- Add the modified digit to the new_num
    ->     SET new_num = new_num + digit * multiplier;
    ->
    ->     -- Move to the next digit
    ->     SET input_num = FLOOR(input_num / 10);
    ->     SET multiplier = multiplier * 10;
    ->   END WHILE;
    ->
    ->   RETURN new_num;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SELECT GenerateNewNumber(123);
+--------------------------+
| GenerateNewNumber(123) |
+--------------------------+
|                      345 |
+--------------------------+
1 row in set (0.00 sec)

mysql> SELECT GenerateNewNumber(32768);
+----------------------------+
| GenerateNewNumber(32768) |
+----------------------------+
|                         -1 |
+----------------------------+
1 row in set (0.00 sec)
```

14. Perfect Number

```
mysql> DELIMITER //
mysql> CREATE FUNCTION FindPerfect(inputNumber INT) RETURNS VARCHAR(20) DETERMINISTIC
    -> BEGIN
    ->     DECLARE sumOfDivisors INT DEFAULT 0;
    ->     DECLARE divisor INT DEFAULT 1;
    ->
    ->     IF inputNumber < 0 OR inputNumber > 32767 THEN
    ->         RETURN 'Invalid Input';
    ->     ELSE
    ->         WHILE divisor < inputNumber DO
    ->             IF inputNumber % divisor = 0 THEN
    ->                 SET sumOfDivisors = sumOfDivisors + divisor;
    ->             END IF;
    ->             SET divisor = divisor + 1;
    ->         END WHILE;
    ->
    ->         IF sumOfDivisors = inputNumber THEN
    ->             RETURN 'yes';
    ->         ELSE
    ->             RETURN 'no';
    ->         END IF;
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT FindPerfect(28);
+-----------------+
| FindPerfect(28) |
+-----------------+
| yes             |
+-----------------+
1 row in set (0.00 sec)
```

15. Product of digits

```
mysql> DELIMITER //
mysql> CREATE FUNCTION productDigits(input_num INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE product INT;
    ->     DECLARE digit INT;
    ->     -- Check for invalid input conditions
    ->     IF input_num < 0 OR input_num > 32767 THEN
    ->         RETURN 'Invalid Input'; -- Return the message as a string
    ->     END IF;
    ->     SET product = 1;
    ->     -- Calculate the product of digits
    ->     WHILE input_num > 0 DO
    ->         SET digit = input_num % 10;
    ->         SET product = product * digit;
    ->         SET input_num = FLOOR(input_num / 10);
    ->     END WHILE;
    ->     RETURN CAST(product AS CHAR);
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> SELECT productDigits(45);
+------------------+
| productDigits(45) |
+------------------+
| 20               |
+------------------+
1 row in set (0.00 sec)
```

**RESULT:** Thus, working with basic PL/SQL programming on MySQL is successfully verified.

| Exp: 9 | Working With PL/SQL Procedures |
|---|---|

**Aim:** To work with PL/SQL Procedures on MySQL.

1) Create a new User

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE CreateUser(
    ->     IN firstName VARCHAR(255),
    ->     IN lastName VARCHAR(255),
    ->     IN gender CHAR(1),
    ->     IN dob DATE,
    ->     IN city VARCHAR(255),
    ->     IN email VARCHAR(255),
    ->     IN accountsHolding TEXT,
    ->     IN contactNumber VARCHAR(15)
    -> )
    -> BEGIN
    ->     INSERT INTO user (First_name, last_name, gender, dob, city, email_id, accounts_holding, contact_number)
    ->     VALUES (firstName, lastName, gender, dob, city, email, accountsHolding, contactNumber);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL CreateUser('Abirami', 'M', 'F', '2005-05-01', 'Salem', 'a@gmail.com', 'Saving', '789');
ERROR 1366 (HY000): Incorrect integer value: 'Saving' for column 'accounts_holding' at row 1
mysql> CALL CreateUser('Abirami', 'M', 'F', '2005-05-01', 'Salem', 'a@gmail.com', 3, 789);
Query OK, 1 row affected (0.01 sec)

mysql> select * from user;
+---------+------------+-----------+--------+------------+-----------+-------------+------------------+----------------+
| user_id | First_name | last_name | gender | dob        | city      | email_id    | accounts_holding | contact_number |
+---------+------------+-----------+--------+------------+-----------+-------------+------------------+----------------+
|     101 | Lokesh     | Kumar     | M      | 2004-01-01 | Chennai   | l@gmail.com |                1 |            123 |
|     202 | Divya      | Murugesan | F      | 2004-04-05 | Bangalore | d@gmail.com |                2 |            890 |
|     303 | Nikhil     | Krishnan  | M      | 2004-05-02 | Coimbatore| s@gmail.com |                3 |            890 |
|     404 | Kavin      | V         | M      | 2004-05-01 | Coimbatore| k@gmail.com |                2 |            908 |
|     505 | Nikitha    | k         | M      | 2005-09-09 | Chennai   | b@gmail.com |                1 |            678 |
|    NULL | Abirami    | M         | F      | 2005-05-01 | Salem     | a@gmail.com |                3 |            789 |
+---------+------------+-----------+--------+------------+-----------+-------------+------------------+----------------+
6 rows in set (0.00 sec)
```

2) Create a new account

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE AddAccount(
    ->     IN accountNumber INT,
    ->     IN holderName VARCHAR(255),
    ->     IN accountType VARCHAR(255),
    ->     IN balance DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     INSERT INTO account (Account_number, Holder_name, Account_type, Balance)
    ->     VALUES (accountNumber, holderName, accountType, balance);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL AddAccount(6, 'Abirami', 'Savings', 1000.00);
Query OK, 1 row affected (0.01 sec)

mysql> select * from account;
+----------------+-------------+--------------+--------------+
| Account_number | Holder_name | Account_type | Balance      |
+----------------+-------------+--------------+--------------+
|              1 | Lokesh      | Savings      |   10000.0000 |
|              2 | Divya       | Savings      |  200000.0000 |
|              3 | Nikhil      | Business     | 4000000.0000 |
|              4 | Kavin       | Current      |    3000.0000 |
|              5 | Nikitha     | Business     |   80000.0000 |
|              6 | Abirami     | Savings      |    1000.0000 |
+----------------+-------------+--------------+--------------+
6 rows in set (0.00 sec)
```

3) Create a new bill

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE AddBill(
    ->     IN userId INT,
    ->     IN name VARCHAR(255),
    ->     IN billType VARCHAR(255),
    ->     IN status VARCHAR(255),
    ->     IN dueDate DATE,
    ->     IN amount DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     INSERT INTO bill (User_id, Name, Bill_type, Status, Due_date, ammount)
    ->     VALUES (userId, name, billType, status, dueDate, amount);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

```
mysql> select * from bill;
+---------+---------+---------+-----------+---------+------------+-----------+
| Bill_id | User_id | Name    | Bill_type | Status  | Due_date   | ammount   |
+---------+---------+---------+-----------+---------+------------+-----------+
|       1 |     101 | Lokesh  | GST       | Pending | 2023-11-09 |   3000.00 |
|       2 |     202 | Divya   | NONGST    | Pending | 2023-09-10 |  10000.00 |
|       3 |     303 | Nikhil  | GST       | Pending | 2023-11-07 | 100000.00 |
|       4 |     404 | Kavin   | GST       | Pending | 2023-11-20 |   1000.00 |
|       5 |     505 | Nikitha | NONGST    | Pending | 2023-03-01 |  20000.00 |
|    NULL |     606 | Abirami | NONGST    | Pending | 2023-11-01 |     50.00 |
+---------+---------+---------+-----------+---------+------------+-----------+
```

4) Create a new Investment

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE CreateInvestment3(
    ->     IN investmentId INT,
    ->     IN userId INT,
    ->     IN amountId INT,
    ->     IN investmentDate DATE,
    ->     IN investmentType VARCHAR(255)
    -> )
    -> BEGIN
    ->     INSERT INTO investment (Investment_id, User_id, Ammount_id, Date_of_investment, Investment_type)
    ->     VALUES (investmentId, userId, amountId, investmentDate, investmentType);
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL CreateInvestment3(6, 606, 107, '2023-11-01', 'Stocks');
Query OK, 1 row affected (0.01 sec)

mysql> select * from investment;
+---------------+---------+------------+--------------------+-----------------+
| Investment_id | User_id | Ammount_id | Date_of_investment | Investment_type |
+---------------+---------+------------+--------------------+-----------------+
|             1 |     101 |        102 | 2023-10-01         | Stock           |
|             2 |     202 |        103 | 2023-10-10         | Mutual Funds    |
|             3 |     303 |        104 | 2023-10-15         | Bonds           |
|             4 |     404 |        105 | 2023-10-20         | Stock           |
|             5 |     505 |        106 | 2023-10-25         | Bonds           |
|             6 |     606 |        107 | 2023-11-01         | Stocks          |
+---------------+---------+------------+--------------------+-----------------+
6 rows in set (0.00 sec)
```

5) Create a new loan

```
mysql> DELIMITER //
mysql>
mysql> CREATE PROCEDURE CreateLoan3(
    ->     IN loadID INT,
    ->     IN accountNo INT,
    ->     IN amount DECIMAL(10, 2),
    ->     IN term INT,
    ->     IN rate DECIMAL(5, 2)
    -> )
    -> BEGIN
    ->     INSERT INTO loan (loan_id, accountno, ammount, term, rate)
    ->     VALUES (loadID, accountNo, amount, term, rate);
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL CreateLoan3(6000, 6, 5000.00, 12, 5.5);
Query OK, 1 row affected (0.01 sec)

mysql> select * from loan;
+---------+-----------+------------+------+------+
| loan_id | accountno | ammount    | term | rate |
+---------+-----------+------------+------+------+
|    1000 |         1 |  600000.00 |    2 | 5.00 |
|    2000 |         2 |  100000.00 |    1 | 2.00 |
|    3000 |         3 | 1000000.00 |    5 | 3.00 |
|    4000 |         4 |   50000.00 |    3 | 2.00 |
|    5000 |         5 |  200000.00 |    2 | 3.00 |
|    6000 |         6 |    5000.00 |   12 | 5.50 |
+---------+-----------+------------+------+------+
6 rows in set (0.00 sec)
```

**RESULT:** Thus, working with PL/SQL Procedures on MySQL is successfully verified.

| Exp: 10 | Working With PL/SQL Functions |
|---|---|

**Aim:** To work with PL/SQL Functions on MySQL

1. Function to Calculate Total Account Balance for a User:

```
mysql> DELIMITER //
mysql> CREATE FUNCTION GetUserTotalBalance1(userId INT) RETURNS DECIMAL(10, 2) DETERMINISTIC
    -> BEGIN
    ->     DECLARE totalBalance DECIMAL(10, 2) DEFAULT 0;
    ->
    ->     SELECT COALESCE(SUM(Balance), 0) INTO totalBalance
    ->     FROM account
    ->     WHERE Account_number = userId;
    ->
    ->     RETURN totalBalance;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SELECT GetUserTotalBalance1(1) AS UserTotalBalance;
+------------------+
| UserTotalBalance |
+------------------+
|         10000.00 |
+------------------+
1 row in set (0.00 sec)
```

2. Function to Calculate Total Bills Amount for a User

```
mysql> DELIMITER //
mysql> CREATE FUNCTION GetUserTotalBillsAmount(userId INT) RETURNS DECIMAL(10, 2) DETERMINIST
    -> BEGIN
    ->     DECLARE totalBillAmount DECIMAL(10, 2) DEFAULT 0;
    ->
    ->     SELECT COALESCE(SUM(ammount), 0) INTO totalBillAmount
    ->     FROM bill
    ->     WHERE User_id = userId;
    ->
    ->     RETURN totalBillAmount;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT GetUserTotalBillsAmount(101) AS UserTotalBillsAmount;
+----------------------+
| UserTotalBillsAmount |
+----------------------+
|              3000.00 |
+----------------------+
1 row in set (0.00 sec)
```

3. Function to Calculate Age from Date of Birth:

```
mysql> DELIMITER //
mysql> CREATE FUNCTION CalculateUserAge(dateOfBirth DATE) RETURNS INT DETERMINISTIC
    -> BEGIN
    ->     DECLARE userAge INT;
    ->
    ->     SET userAge = TIMESTAMPDIFF(YEAR, dateOfBirth, CURDATE());
    ->
    ->     RETURN userAge;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT CalculateUserAge('2004-01-01') AS UserAge;
+---------+
| UserAge |
+---------+
|      19 |
+---------+
1 row in set (0.00 sec)
```

4. Function to Retrieve User's Full Name

```
mysql> DELIMITER //
mysql> CREATE FUNCTION GetUserFullName(userId INT) RETURNS VARCHAR(255) DETERMINISTIC
    -> BEGIN
    ->     DECLARE userFullName VARCHAR(255);
    ->
    ->     SELECT CONCAT(First_name, ' ', last_name) INTO userFullName
    ->     FROM user
    ->     WHERE user_id = userId;
    ->
    ->     RETURN userFullName;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT GetUserFullName(303) AS FullName;
+-----------------+
| FullName        |
+-----------------+
| Nikhil Krishnan |
+-----------------+
1 row in set (0.00 sec)
```

5. **Function to Check if a User has a Specific Account Type**

```
mysql> DELIMITER //
mysql> CREATE FUNCTION UserHasAccountType(userId INT, accountTypeToCheck CHAR(20)) RETURNS BOOLEAN DETERMINISTIC
    -> BEGIN
    ->     DECLARE hasAccountType BOOLEAN;
    ->
    ->     SELECT COUNT(*) INTO hasAccountType
    ->     FROM account
    ->     WHERE Account_number = userId AND Account_type = accountTypeToCheck;
    ->
    ->     RETURN hasAccountType > 0;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT UserHasAccountType(101, 'Savings') AS HasSavingsAccount;
+-------------------+
| HasSavingsAccount |
+-------------------+
|                 0 |
+-------------------+
1 row in set (0.00 sec)
```

**RESULT:** Thus, working with PL/SQL Functions on MySQL is successfully verified.

| Exp: 11 | Working With PL/SQL Cursors |
|---|---|

**Aim:** To work with PL/SQL Cursors on MySQL.

1. Cursor to Retrieve User's Bills

```
mysql>  DELIMITER //
mysql> CREATE PROCEDURE GetUserBills1(IN userId INT)
    -> BEGIN
    ->     DECLARE done INT DEFAULT 0;
    ->     DECLARE billId INT;
    ->     DECLARE billName VARCHAR(255);
    ->     DECLARE billType VARCHAR(255);
    ->     DECLARE billStatus VARCHAR(255);
    ->     DECLARE dueDate DATE;
    ->     DECLARE amount DECIMAL(10, 2);
    ->
    ->     DECLARE cur CURSOR FOR
    ->     SELECT Bill_id, Name, Bill_type, Status, Due_date, ammount
    ->     FROM bill
    ->     WHERE User_id = userId;
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    ->
    ->     OPEN cur;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur INTO billId, billName, billType, billStatus, dueDate, amount;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Process the retrieved data here
    ->         SELECT billId, billName, billType, billStatus, dueDate, amount;
    ->     END LOOP;
    ->
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL GetUserBills1(101);
+--------+----------+----------+------------+------------+---------+
| billId | billName | billType | billStatus | dueDate    | amount  |
+--------+----------+----------+------------+------------+---------+
|      1 | Lokesh   | GST      | Pending    | 2023-11-09 | 3000.00 |
+--------+----------+----------+------------+------------+---------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

2. Cursor to Calculate Total Balance

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE CalculateTotalBalance()
    -> BEGIN
    ->     DECLARE done INT DEFAULT 0;
    ->     DECLARE totalBalance DECIMAL(10, 2) DEFAULT 0;
    ->     DECLARE accountBalance DECIMAL(10, 2);
    ->
    ->     DECLARE cur CURSOR FOR
    ->     SELECT Balance FROM account;
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    ->
    ->     OPEN cur;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur INTO accountBalance;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Calculate total balance
    ->         SET totalBalance = totalBalance + accountBalance;
    ->     END LOOP;
    ->
    ->     CLOSE cur;
    ->
    ->     -- Return the total balance
    ->     SELECT totalBalance;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL CalculateTotalBalance();
+--------------+
| totalBalance |
+--------------+
|   4294000.00 |
+--------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

3. Cursor to List Users with Overdue Bills

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE ListUsersWithOverdueBills1()
    -> BEGIN
    ->     DECLARE done INT DEFAULT 0;
    ->     DECLARE userId INT;
    ->     DECLARE firstName VARCHAR(255);
    ->     DECLARE lastName VARCHAR(255);
    ->
    ->     DECLARE cur CURSOR FOR
    ->     SELECT u.user_id, u.First_name, u.Last_name
    ->     FROM user u
    ->     JOIN bill b ON u.user_id = b.User_id
    ->     WHERE b.Status = 'Pending' AND b.Due_date < CURDATE();
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    ->
    ->     OPEN cur;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur INTO userId, firstName, lastName;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Process the retrieved user data
    ->         SELECT userId, firstName, lastName;
    ->     END LOOP;
    ->
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL ListUsersWithOverdueBills1();
+--------+-----------+-----------+
| userId | firstName | lastName  |
+--------+-----------+-----------+
|    202 | Divya     | Murugesan |
+--------+-----------+-----------+
1 row in set (0.00 sec)

+--------+-----------+----------+
| userId | firstName | lastName |
+--------+-----------+----------+
|    505 | Nikitha   | k        |
+--------+-----------+----------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

4. Cursor to Retrieve Investment Details

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetUserInvestments(IN userId INT)
    -> BEGIN
    ->     DECLARE done INT DEFAULT 0;
    ->     DECLARE investmentId INT;
    ->     DECLARE investmentType VARCHAR(255);
    ->     DECLARE investmentDate DATE;
    ->
    ->     DECLARE cur CURSOR FOR
    ->     SELECT Investment_id, Investment_type, Date_of_investment
    ->     FROM Investment
    ->     WHERE User_id = userId;
    ->
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    ->
    ->     OPEN cur;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur INTO investmentId, investmentType, investmentDate;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Process the retrieved investment data here
    ->         SELECT investmentId, investmentType, investmentDate;
    ->     END LOOP;
    ->
    ->     CLOSE cur;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL GetUserInvestments(303);
+--------------+----------------+----------------+
| investmentId | investmentType | investmentDate |
+--------------+----------------+----------------+
|            3 | Bonds          | 2023-10-15     |
+--------------+----------------+----------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

**RESULT:** Thus, working with PL/SQL Cursors on MySQL is successfully verified.

| Exp: 12 | Working With PL/SQL Triggers |
|---|---|

**Aim:** To work with PL/SQL Triggers on MySQL.

1. Trigger to Update User Account Balance After Bill Payment:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER update_balance_after_payment
    -> AFTER INSERT ON bill
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE account
    ->     SET Balance = Balance - NEW.ammount
    ->     WHERE Account_number = NEW.User_id;
    -> END;
    -> //
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SELECT * FROM account WHERE Account_number = 1;
+----------------+-------------+--------------+-----------+
| Account_number | Holder_name | Account_type | Balance   |
+----------------+-------------+--------------+-----------+
|              1 | Lokesh      | Savings      | 9950.0000 |
+----------------+-------------+--------------+-----------+
1 row in set (0.00 sec)
```

2. Trigger to Enforce a Maximum Investment Limit:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER enforce_investment_limit
    -> BEFORE INSERT ON investment
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE max_investment DECIMAL(10, 2) DEFAULT 10000.00;
    ->     IF NEW.Investment_type = 'Stocks' AND NEW.Ammount_id > max_investment THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Investment exceeds maximum limit';
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;
mysql> INSERT INTO investment (Investment_type, Ammount_id) VALUES ('Stocks', 15000.00);
ERROR 1644 (45000): Investment exceeds maximum limit
```

3. Trigger to Calculate and Update Loan Repayments:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER calculate_loan_repayment1
    -> BEFORE INSERT ON loan
    -> FOR EACH ROW
    -> BEGIN
    ->     SET NEW.ammount =NEW.loan_id+NEW.accountno+ NEW.ammount + (NEW.ammount * NEW.rate / 100) * NEW.term;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO loan (loan_id,accountno,ammount, rate, term) VALUES (7000,7,10000, 5, 2);
    -> delemitter;
    -> INSERT INTO loan (loan_id,accountno,ammount, rate, term) VALUES (7000,7,10000, 5, 2);
    -> //
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from loan;
    -> //
+---------+-----------+------------+------+------+
| loan_id | accountno | ammount    | term | rate |
+---------+-----------+------------+------+------+
|    1000 |         1 |  600000.00 |    2 | 5.00 |
|    2000 |         2 |  100000.00 |    1 | 2.00 |
|    3000 |         3 | 1000000.00 |    5 | 3.00 |
|    4000 |         4 |   50000.00 |    3 | 2.00 |
|    5000 |         5 |  200000.00 |    2 | 3.00 |
|    6000 |         6 |    5000.00 |   12 | 5.50 |
|    7000 |         7 |   19107.00 |    2 | 5.00 |
+---------+-----------+------------+------+------+
7 rows in set (0.00 sec)
```

4. Trigger to Prevent Deletion of User Accounts with Active Bills:

```
mysql> DELIMITER //
mysql> CREATE TRIGGER prevent_account_deletion1
    -> BEFORE DELETE ON user
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE unpaid_bills INT;
    ->     SELECT COUNT(*) INTO unpaid_bills
    ->     FROM bill
    ->     WHERE User_id = OLD.user_id AND Status = 'Pending';
    ->     IF unpaid_bills > 0 THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Cannot delete user with unpaid bills';
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
mysql> DELETE FROM user WHERE user_id = 101;
ERROR 1644 (45000): Cannot delete user with unpaid bills
```

**RESULT:** Thus, working with PL/SQL Triggers on MySQL is successfully verified.

| Exp: 13 | Project Report |
|---|---|

## Project Report: Fitness Tracker

1. Introduction:

In the era of modern technology, staying fit and healthy is a priority for many individuals. To aid in this endeavor, the development of a Fitness Tracker application has been undertaken. This application serves as a tool to monitor and track fitness activities, providing users with valuable insights into their exercise routines and progress.

2. Objective:

The primary objective of the Fitness Tracker project is to create a comprehensive platform that allows users to:

Log in securely to their accounts
Record exercise activities along with duration
Access a database of exercises categorized by body parts
Retrieve and analyze their fitness data
3. Technologies Used:

The project utilizes the following technologies:

Frontend: HTML, CSS, JavaScript
Backend: Node.js, Express.js, MySQL
Database: MySQL
4. Features:

The Fitness Tracker application comprises the following features:

User Authentication: Users can securely log in to their accounts using their credentials.
Exercise Logging: Users can record their exercise activities, specifying the type of exercise and its duration.
Exercise Database: The application contains a database of exercises categorized by body parts, allowing users to select exercises easily.
Data Storage: User account details, exercise logs, and exercise database are stored in separate tables within the MySQL database.
Data Analysis: Users can view their exercise history and track their progress over time.
5. Database Structure:

6. Implementation:

Frontend: The frontend of the application is developed using HTML, CSS, and JavaScript. It includes user interface elements for logging in, recording exercises, and viewing exercise history.

Backend: The backend is implemented using Node.js and Express.js framework. It handles user authentication, database operations, and serves API endpoints for frontend interactions.

Database Integration: MySQL database is integrated with the backend to store user account details, exercise logs, and exercise database.

Security Measures: Proper measures are implemented to ensure the security of user data, including encryption of passwords and input validation.

7. Conclusion:

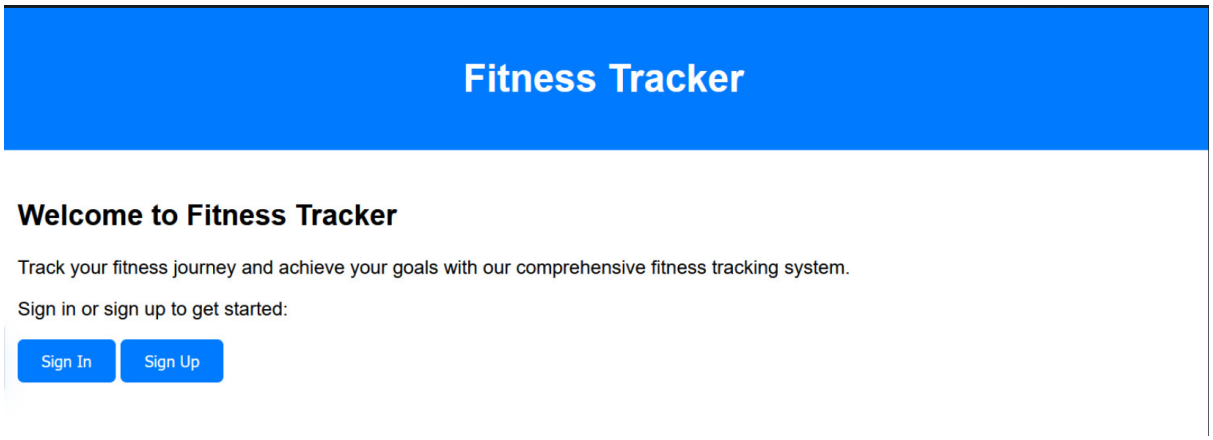
The Fitness Tracker project aims to provide users with a user-friendly platform to monitor and track their fitness activities effectively. By integrating frontend, backend, and database functionalities, the application offers a comprehensive solution for individuals striving towards their fitness goals.

8. Future Enhancements:

Implementation of data visualization tools for better analysis of fitness data.
Integration with wearable fitness devices for real-time data syncing.
Incorporation of social features to enable users to connect and compete with friends.
Expansion of exercise database with additional exercises and categorizations.

## Conclusion:

**The Fitness Tracker project serves as an innovative tool for individuals seeking to enhance their fitness routines. By combining technology with health insights, this application aims to motivate users to achieve their fitness goals effectively.**



# Sign In

**Email:**

**Password:**

Submit

# Fitness Tracker Application

ID

Enter ID

Name

Enter Name

Age

Enter Age

Sex

Enter Sex

Height (cm)

Enter Height

Weight (kg)

Enter Weight

Submit