Dashboard Visualizations with Analytical Query Explanation.

1. Top 5 Players By Win Ratio

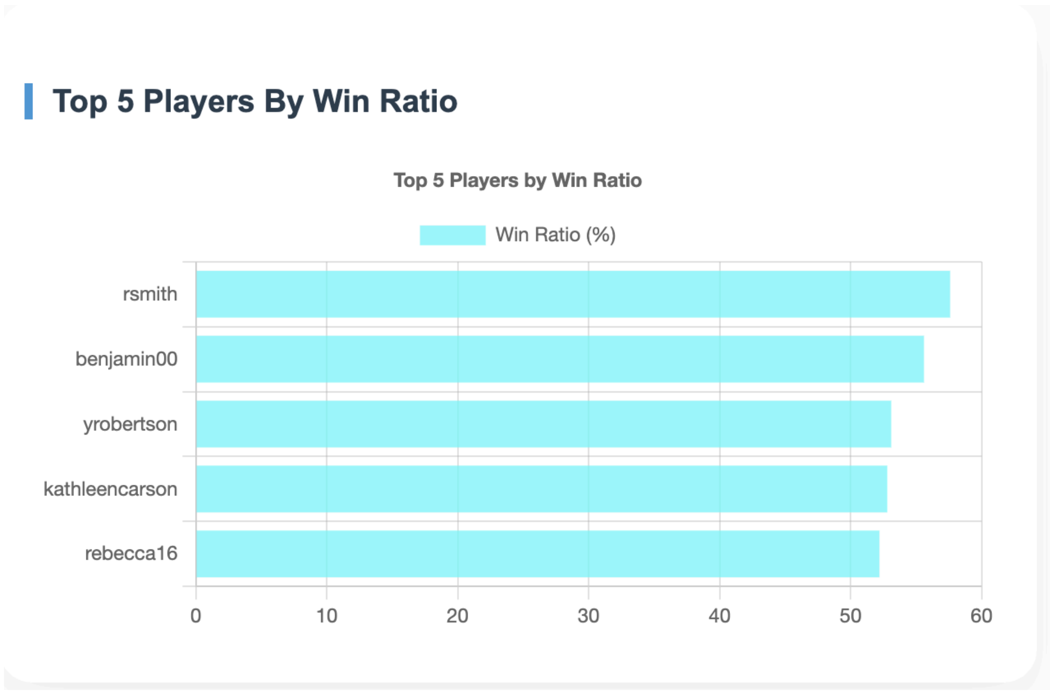

Chart Type: Horizontal Bar Chart
Metric: Win Ratio (%)

This chart showcases the top five players who have achieved the highest win ratio, calculated as:

```
SELECT
        PlayerID,
        Username,
        Wins,
        Losses,
        Draws,
        Withdraw,
        ROUND(
                CASE
                WHEN (Wins + Losses + Draws + Withdraw) = 0 THEN 0
                ELSE (Wins / (Wins + Losses + Draws + Withdraw)) * 100
```

```
                    END, 2
        ) AS WinRate
FROM
    Player
ORDER BY
    WinRate DESC
LIMIT 5;
```

This provides insight into which players perform most efficiently, regardless of the number of games played. A high win ratio implies consistent performance.

2. Top 5 Players by Longest Total Playtime

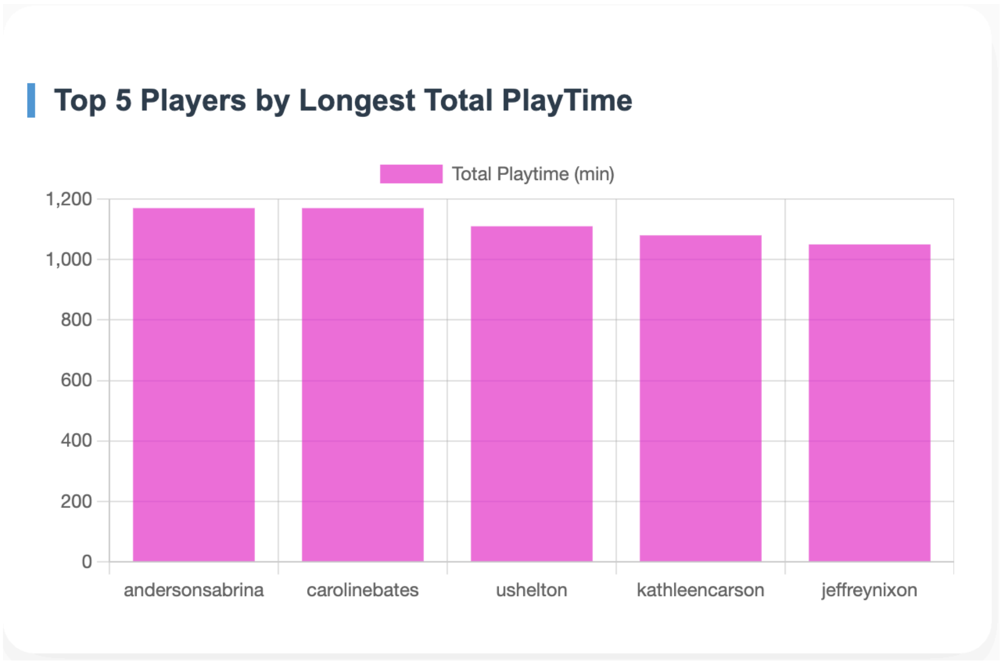

Chart Type: Vertical Bar Chart
Metric: Total Playtime (in minutes)

```
SELECT

    sub.PlayerID,
    sub.Username,
    sub.Week_Range,
    SUM(sub.Playtime_Minutes) AS Total_Playtime_Minutes
FROM (
    SELECT
        p.PlayerID,
        p.Username,
        CONCAT(
            DATE_FORMAT(DATE_SUB(g.Game_start, INTERVAL
        WEEKDAY(g.Game_start) DAY), '%d/%m/%Y'),
            ' - ',
            DATE_FORMAT(DATE_ADD(g.Game_start, INTERVAL (6 -
        WEEKDAY(g.Game_start)) DAY), '%d/%m/%Y')
        ) AS Week_Range,
    TIMESTAMPDIFF(MINUTE, g.Game_start, g.Game_end) AS Playtime_Minutes
    FROM Player p
```

```
        JOIN PlayerGameSession pgs ON p.PlayerID = pgs.PlayerID
        JOIN Game g ON pgs.GameID = g.GameID
        WHERE g.Game_end IS NOT NULL
) AS sub
GROUP BY
        sub.PlayerID,
        sub.Username,
        sub.Week_Range
ORDER BY Total_Playtime_Minutes DESC
LIMIT 5;
```

This chart reveals the most engaged players in terms of time spent playing. It highlights dedication and frequent participation in game sessions.

3. Top 5 Players By Win Rate – Win / Loss / Draw Stats

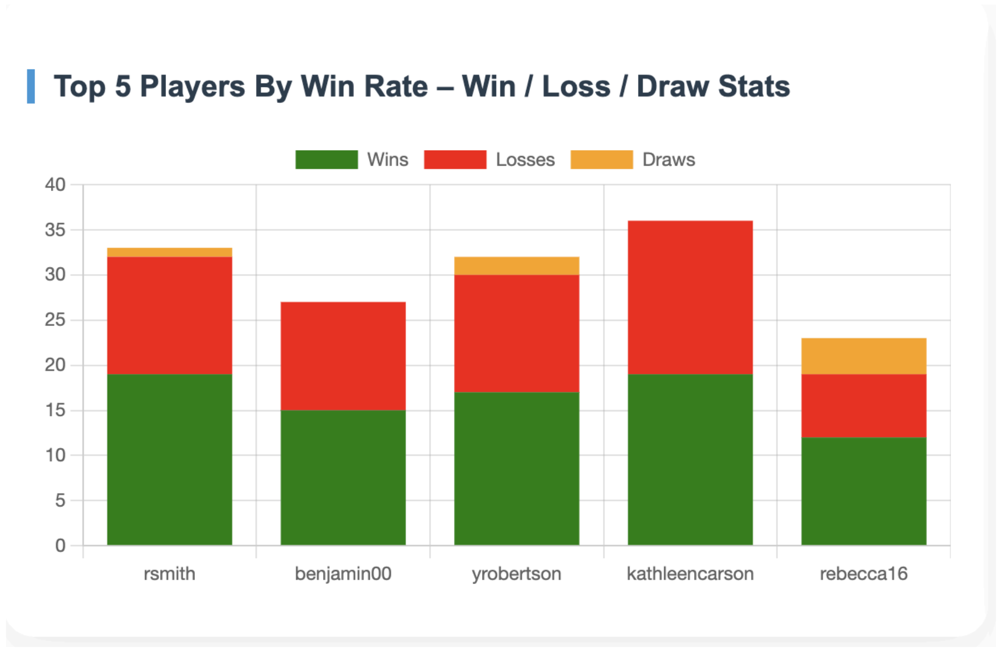

Chart Type: Stacked Vertical Bar Chart
Metrics: Wins, Losses, Draws

```
SELECT
        Username AS playerName,
        Wins,
        Losses,
        Draws,
        (Wins / (Wins + Losses + Draws + 0.0)) AS win_ratio
FROM Player
ORDER BY win_ratio DESC
LIMIT 5;
```

This visualization provides a breakdown of the top 5 players' outcomes in terms of wins, losses, and draws. It supplements the win ratio chart by offering context on the number of games and the spread of results.

4. Top 5 Players by Most Unlocked Achievements

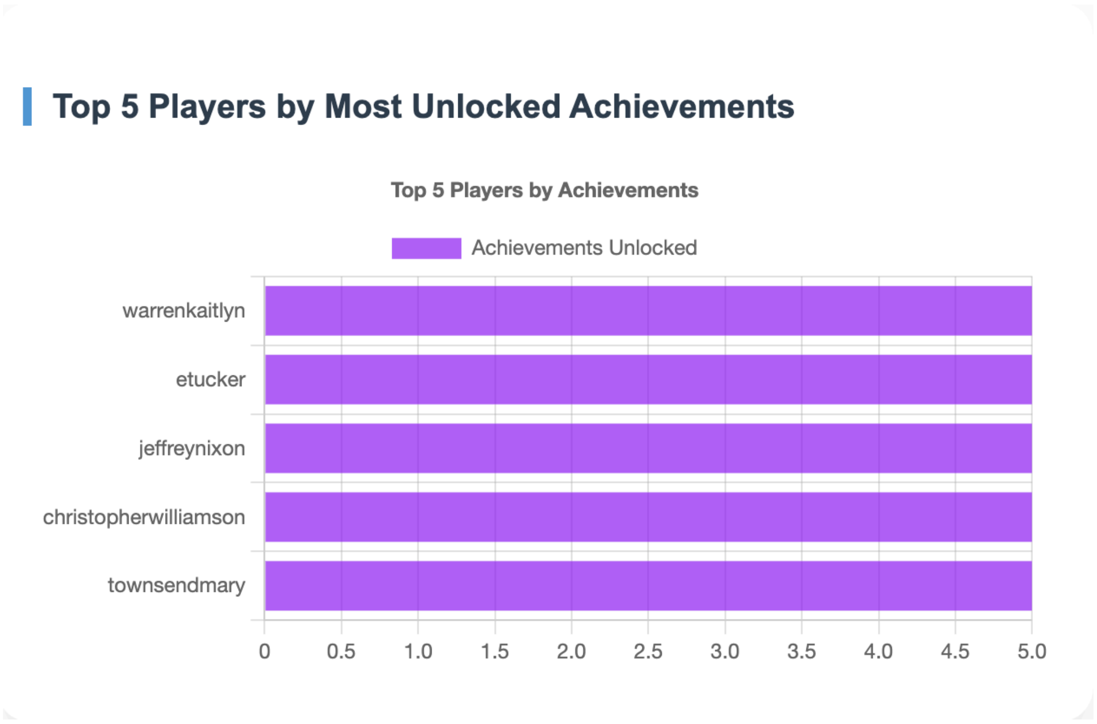
Top 5 Players by Most Unlocked Achievements

Chart Type: Horizontal Bar Chart
Metric: Count of Achievements

```
SELECT
        p.Username,
        PlayerPlayTime.playtime,
        achievements
FROM (
        SELECT
                pgs.PlayerID,
                SEC_TO_TIME(SUM(TIMESTAMPDIFF(SECOND, g.Game_start, g.Game_end)))
        AS playtime
        FROM PlayerGameSession AS pgs
        LEFT JOIN Game AS g ON pgs.GameID = g.GameID
        GROUP BY pgs.PlayerID
) AS PlayerPlayTime
LEFT JOIN Player AS p ON PlayerPlayTime.PlayerID = p.PlayerID
LEFT JOIN (
        SELECT
                pa.PlayerID,
                COUNT(pa.AchievementID) AS achievements
                FROM PlayerAchievement AS pa
```

```
                    GROUP BY pa.PlayerID
          ) AS achieves ON p.PlayerID = achieves.PlayerID
ORDER BY achievements DESC
LIMIT 5;
```

This chart identifies players who have unlocked the most achievements. These players are likely to be more experienced or goal-oriented within the system.

5. Paid vs Unpaid Players

Paid vs Unpaid Players

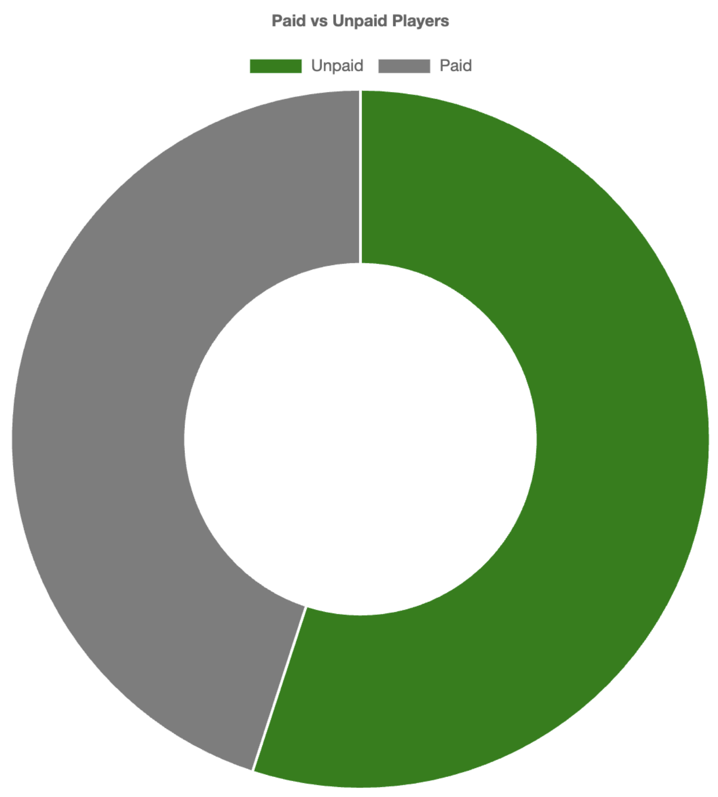

Chart Type: Donut Chart
Metric: Player Status

```
SELECT
	CASE WHEN pp.PlayerID IS NOT NULL THEN 'Paid' ELSE 'Unpaid' END AS
PaymentStatus,
	COUNT(p.PlayerID) AS Count
FROM Player p
LEFT JOIN PaidPlayer pp ON p.PlayerID = pp.PlayerID
GROUP BY PaymentStatus;
```

This visualization helps assess monetization and feature access within the player base. A high number of paid users indicates effective feature conversion.

6. Player Registration Trend

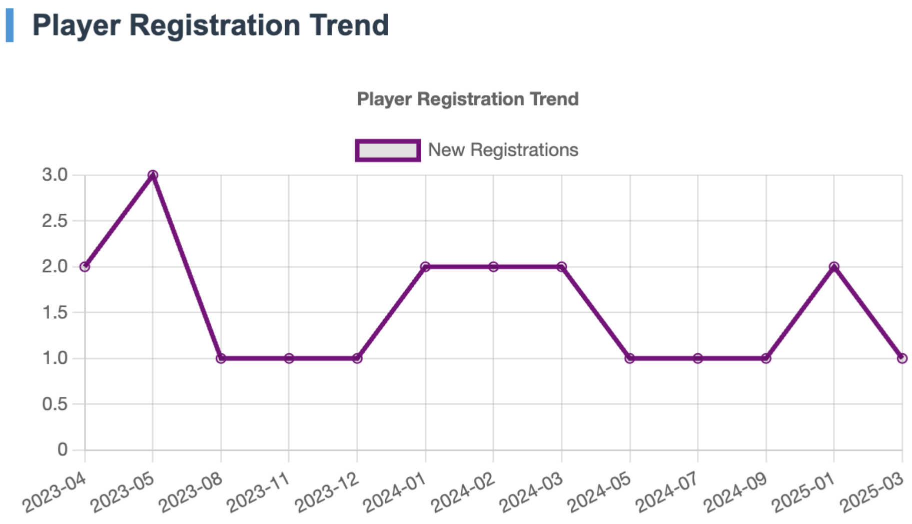

Chart Type: Line Chart
Metric: Number of Registrations per Month

```
SELECT
        DATE_FORMAT(registration_date, '%Y-%m') AS Month,
        COUNT(*) AS Registrations
FROM Player
GROUP BY Month
ORDER BY Month;
```

This chart tracks how many players registered each month. It’s useful for identifying trends, marketing campaign impact, and seasonal user growth patterns.

List of Analytical Queries: Textual and SQL

app/features/delete_player.php

SQL:

1. Fetch Player Details

```
SELECT * FROM Player WHERE PlayerID = $id;
```

2. Delete Sessions Related to the Player

```
DELETE FROM Session
WHERE SessionID IN (
        SELECT * FROM (
        SELECT s.SessionID
        FROM `Session` AS s
        RIGHT JOIN PlayerGameSession AS pgs ON pgs.SessionID = s.SessionID
        WHERE pgs.PlayerID = $id
        ) AS tmp
);
```

3. Delete the Player

```
DELETE FROM Player WHERE PlayerID = $id;
```

Textual Explanation:

1. The script starts by checking if a player ID (id) is provided via the URL ($_GET['id']).
2. If the ID exists:
    - It retrieves the player's details from the Player table using the ID.
    - If the player exists, it continues with deletion.
3. Before deleting the player:
    - It deletes all Session records related to the player that won't be deleted automatically by cascading foreign key constraints. These are fetched via a right join between Session and PlayerGameSession on SessionID.
4. Then, it deletes the player from the Player table.
5. Success or error messages are displayed based on the outcomes of each deletion.
6. Finally, a link is shown to go back to the player list.

app/features/get_player_achievement.php

SQL:

```
SELECT
	PlayerID,
	Username,
	Wins,
	Losses,
	Draws,
	Withdraw,
	ROUND(
	CASE
	WHEN (Wins + Losses + Draws + Withdraw) = 0 THEN 0
	ELSE (Wins / (Wins + Losses + Draws + Withdraw)) * 100
	END, 2
	) AS WinRate
FROM
	Player
ORDER BY
	WinRate DESC, Wins DESC;
```

Textual Explanation:

1. The script runs a SQL query that fetches player stats (`PlayerID`, `Username`, `Wins`, `Losses`, `Draws`, `Withdraw`) from the `Player` table.
2. It also calculates each player's Win Rate, the percentage of wins out of total games played, rounded to 2 decimal places. The formula is: WinRate = [Wins / (Wins + Losses + Draws + Withdraw)] x 100. If the total number of games is zero, the win rate is set to `0`.
3. The players are ordered by `WinRate` descending, and then by `Wins` descending as a tiebreaker.

app/features/get_players_achievement_results.php

SQL:

```
SELECT
        a.name AS AchievementName,
        pa.achievement_datetime,
        s.SessionID,
        s.Session_start,
        s.Session_end
FROM
        PlayerAchievement pa
JOIN
        Achievements a ON pa.AchievementID = a.AchievementID
LEFT JOIN Session s ON
        s.SessionID = (
        SELECT s1.SessionID
        FROM Session s1
        WHERE s1.Session_start <= pa.achievement_datetime
        AND s1.Session_end >= pa.achievement_datetime
        ORDER BY s1.Session_start ASC
        LIMIT 1
        )
WHERE
        pa.PlayerID = ?
ORDER BY
        pa.achievement_datetime ASC;
```

Textual Explanation:

This SQL query retrieves a list of achievements unlocked by a specific player, along with the session (if any) that was happening at the time the achievement was earned:

- `a.name`: The name of the achievement from the `Achievements` table.
- `pa.achievement_datetime`: The timestamp when the player unlocked the achievement.
- `s.SessionID`, `s.Session_start`, `s.Session_end`: Information about the earliest session that overlaps with the achievement timestamp.

To determine the matching session:

- A subquery finds the first session (by start time) where:
    - The session started on or before the achievement's unlock time.
    - The session ended on or after that time, meaning the achievement was unlocked during the session.

Finally, the results are filtered by the specific player (`PlayerID = ?`) and ordered chronologically by the achievement unlock time.

app/features/get_players_stat.php

SQL:

```
SELECT
        p.Username,
        PlayerPlayTime.playtime,
        achievements
FROM (
        SELECT
        pgs.PlayerID,
        SEC_TO_TIME(SUM(TIMESTAMPDIFF(SECOND, g.Game_start, g.Game_end))) AS
playtime
        FROM
        PlayerGameSession AS pgs
        LEFT JOIN
        Game AS g ON pgs.GameID = g.GameID
        GROUP BY
        pgs.PlayerID
) AS PlayerPlayTime
LEFT JOIN
        Player AS p ON PlayerPlayTime.PlayerID = p.PlayerID
LEFT JOIN (
        SELECT
        pa.PlayerID,
        COUNT(pa.AchievementID) AS achievements
        FROM
        PlayerAchievement AS pa
        GROUP BY
        pa.PlayerID
) AS achieves ON p.PlayerID = achieves.PlayerID
ORDER BY
        [sort_column] [ASC|DESC];
```

Note: `[sort_column]` is dynamically replaced by either `Username`, `playtime`, or `achievements`, and `[ASC|DESC]` depends on the selected sort order from the user input.

Textual Explanation:

This query fetches a summary of player statistics with dynamic sorting options.
Playtime Subquery (`PlayerPlayTime`):

- Joins `PlayerGameSession` (`pgs`) with `Game` (`g`) using `GameID`.
- For each `PlayerID`, calculates the total duration of all games played.

- The total seconds are summed and converted into a readable `HH:MM:SS` format using `SEC_TO_TIME(...)`.
- This result is grouped by each player.

Achievements Subquery (`achieves`):

- Counts the number of achievements (`AchievementID`) each player has in the `PlayerAchievement` table.
- This count is grouped by `PlayerID`.

Joining Everything Together:

- The aggregated playtime (`PlayerPlayTime`) and achievement count (`achieves`) are **LEFT JOINed** with the `Player` table on `PlayerID`.
- This ensures that even players who don't have game sessions or achievements still appear in the results (with NULL or default values).
- **ORDER BY** dynamically sorts the final result based on user input (`Username`, `playtime`, or `achievements`) and sort direction (`ASC` or `DESC`)

app/features/get_stored_procedures_options.php

SQL:

```
SELECT
        ProcedureText
FROM
        StoredProcedures
ORDER BY
        ProcedureID;
```

Textual Explanation:

This SQL query retrieves a list of all stored procedures from the StoredProcedures table:

- ProcedureText: The text of each stored procedure is selected.
- The results are ordered by ProcedureID to ensure a consistent and logical sequence, reflecting the order in which the procedures were created/listed.

app/features/playtime_per_week.php

SQL:

```
SELECT
        sub.PlayerID,
        sub.Username,
        sub.Week_Range,
        SUM(sub.Playtime_Minutes) AS Total_Playtime_Minutes
FROM (
        SELECT
        p.PlayerID,
        p.Username,
        CONCAT(
        DATE_FORMAT(DATE_SUB(g.Game_start, INTERVAL WEEKDAY(g.Game_start) DAY),
'%d/%m/%Y'),
        ' - ',
        DATE_FORMAT(DATE_ADD(g.Game_start, INTERVAL (6 - WEEKDAY(g.Game_start))
DAY), '%d/%m/%Y')
        ) AS Week_Range,
        TIMESTAMPDIFF(MINUTE, g.Game_start, g.Game_end) AS Playtime_Minutes
        FROM
        Player p
        JOIN
        PlayerGameSession pgs ON p.PlayerID = pgs.PlayerID
        JOIN
        Game g ON pgs.GameID = g.GameID
        WHERE
        g.Game_end IS NOT NULL
) AS sub
GROUP BY
        sub.PlayerID,
        sub.Username,
        sub.Week_Range
ORDER BY
        [sort_column] [ASC|DESC];
```

Textual Explanation:

This SQL query calculates the total amount of time each player has spent in games, grouped by week. It starts by joining the `Player`, `PlayerGameSession`, and `Game` tables to access each player's game sessions. For each session, it determines which calendar week the game belongs to by calculating the Monday and Sunday dates surrounding the game's start time. This is done using `DATE_SUB` and

`DATE_ADD` with `WEEKDAY()` to align the session with the standard Monday-to-Sunday weekly range. The result is a readable week label like “11/03/2024 - 17/03/2024”.

It then calculates the duration of each game session in minutes using `TIMESTAMPDIFF(MINUTE, g.Game_start, g.Game_end)` and filters out any sessions that don’t have an end time (likely incomplete or in-progress games). This processed data is wrapped as a subquery named `sub`.

The outer query groups this subquery data by `PlayerID`, `Username`, and the computed `Week_Range`, and then sums up the playtime for each group. This provides the total playtime per player per week. The final results are sorted dynamically based on user input—either by `PlayerID`, `Username`, `Week_Range`, or `Total_Playtime_Minutes` in ascending or descending order.

app/features/store_procedure_text.php

SQL:

INSERT INTO StoredProcedures (ProcedureText) VALUES (?);

Textual Explanation:

This script inserts a new stored procedure into the `StoredProcedures` table.

app/features/top5.php

SQL:

```
SELECT
        PlayerID,
        Username,
        Wins,
        Losses,
        Draws,
        Withdraw,
        (Wins / (Wins + Losses + Draws + 0.0)) AS win_ratio
FROM
        Player
ORDER BY
        win_ratio DESC
LIMIT 5;
```

Textual Explanation:

This SQL query retrieves the top 5 players from the `Player` table, ranked by their win ratio. It selects each player's `PlayerID`, `Username`, and game statistics including `Wins`, `Losses`, `Draws`, and `Withdraw`. The key calculated field is `win_ratio`, which is computed as the number of wins divided by the total number of games played (i.e., `Wins + Losses + Draws`). To avoid integer division and ensure accurate results, `0.0` is added to the denominator, forcing a float division. The query then sorts the results in descending order based on the win ratio and limits the output to the top 5 players.

app/features/update_player.php

SQL:

```
UPDATE Player
SET Username = '$username',
        email_address = '$email'
WHERE PlayerID = $id;
```

Textual Explanation:

1. This script is triggered only when the form in edit_player.php is submitted via POST.
2. It reads the following from the submitted form:
    - PlayerID (hidden input field)
    - username (new or updated username)
    - email (new or updated email)
3. It uses real_escape_string() to escape special characters in the username and email inputs to reduce the risk of SQL injection.
4. It constructs and executes an UPDATE SQL query that:
    - Changes the Username and email_address for the player matching the given PlayerID.
5. If the update is successful, it prints a success message with a link back to the player list.
6. If something goes wrong with the SQL execution, it shows an error message.
7. If the page wasn't accessed through a POST request, it shows an "Invalid request" message.