



# PLAYER PERFORMANCE AND GAME DYNAMICS ANALYSIS - USING SQL

BY SHUBHAM MATAGHARE

# PROJECT GOAL :

- ANALYZE PLAYER PERFORMANCE:** EXTRACT KEY PERFORMANCE METRICS FOR PLAYERS
- DEVICE AND DIFFICULTY ANALYSIS:** COMPARE PLAYER PERFORMANCE ACROSS DEVICES AND DIFFICULTY LEVELS
- GAME LEVEL INSIGHTS:** CALCULATE AVERAGE PERFORMANCE METRICS FOR DIFFERENT GAME LEVELS
- IDENTIFY TRENDS AND PATTERNS:** TRACK CUMULATIVE PLAYER PERFORMANCE OVER TIME
- PLAYER BEHAVIOR ANALYSIS:** EXAMINE PLAYER ENGAGEMENT AND BEHAVIOR PATTERNS
- OPTIMIZE GAME EXPERIENCE:** SUGGEST IMPROVEMENTS BASED ON PERFORMANCE INSIGHTS

# KEY QUESTIONS:

1. EXTRACT 'P\_ID', 'DEV\_ID', 'PNAME', AND 'DIFFICULTY\_LEVEL' OF ALL PLAYERS AT LEVEL 0
2. FIND THE TOTAL NUMBER OF STAGES CROSSED AT EACH DIFFICULTY LEVEL FOR LEVEL 2 WITH PLAYERS
3. EXTRACT 'P\_ID' AND THE TOTAL NUMBER OF UNIQUE DATES FOR THOSE PLAYERS WHO HAVE PLAYED GAMES ON MULTIPLE DAYS
4. FIND 'P\_ID' AND LEVELWISE SUM OF 'KILL\_COUNTS' WHERE 'KILL\_COUNT' IS GREATER THAN THE AVERAGE KILL COUNT FOR MEDIUM DIFFICULTY
5. FIND 'LEVEL' AND ITS CORRESPONDING 'LEVEL\_CODE' WISE SUM OF LIVES EARNED, EXCLUDING LEVEL 0. ARRANGE IN ASCENDING ORDER OF LEVEL
6. FIND THE TOP 3 SCORES BASED ON EACH 'DEV\_ID' AND RANK THEM IN INCREASING ORDER USING 'ROW\_NUMBER'. DISPLAY THE DIFFICULTY AS WELL
7. FIND THE 'FIRST\_LOGIN' DATETIME FOR EACH DEVICE ID
8. FIND THE DEVICE ID THAT IS FIRST LOGGED IN (BASED ON 'START\_DATETIME') FOR EACH PLAYER ('P\_ID'). OUTPUT SHOULD CONTAIN PLAYER ID, DEVICE ID, AND FIRST LOGIN DATETIME
9. EXTRACT THE TOP 3 HIGHEST SUMS OF SCORES FOR EACH 'DEV\_ID' AND THE CORRESPONDING 'P\_ID'
10. FIND THE CUMULATIVE SUM OF STAGES CROSSED OVER 'START\_DATETIME' FOR EACH 'P\_ID', EXCLUDING THE MOST RECENT 'START\_DATETIME'

# **DATASET DESCRIPTION**

## PLAYER DETAILS TABLE:

- `P\_ID`: PLAYER ID
- `PNAME`: PLAYER NAME
- `L1\_STATUS`: LEVEL 1 STATUS
- `L2\_STATUS`: LEVEL 2 STATUS
- `L1\_CODE`: SYSTEMGENERATED LEVEL 1 CODE
- `L2\_CODE`: SYSTEMGENERATED LEVEL 2 CODE

## LEVEL DETAILS TABLE:

- `P\_ID`: PLAYER ID
- `DEV\_ID`: DEVICE ID
- `START\_TIME`: START TIME
- `STAGES\_CROSSED`: STAGES CROSSED
- `LEVEL`: GAME LEVEL
- `DIFFICULTY`: DIFFICULTY LEVEL
- `KILL\_COUNT`: KILL COUNT
- `HEADSHOTS\_COUNT`: HEADSHOTS COUNT
- `SCORE`: PLAYER SCORE
- `LIVES\_EARNED`: EXTRA LIVES EARNED

The image features a light gray background with a subtle gradient. In the top-left and bottom-right corners, there are several realistic-looking water droplets of various sizes, some overlapping. The droplets have highlights and shadows, giving them a three-dimensional appearance. Centered on the page is the text "QUESTIONS AND SQL QUERIES" in a bold, black, sans-serif font.

# QUESTIONS AND SQL QUERIES

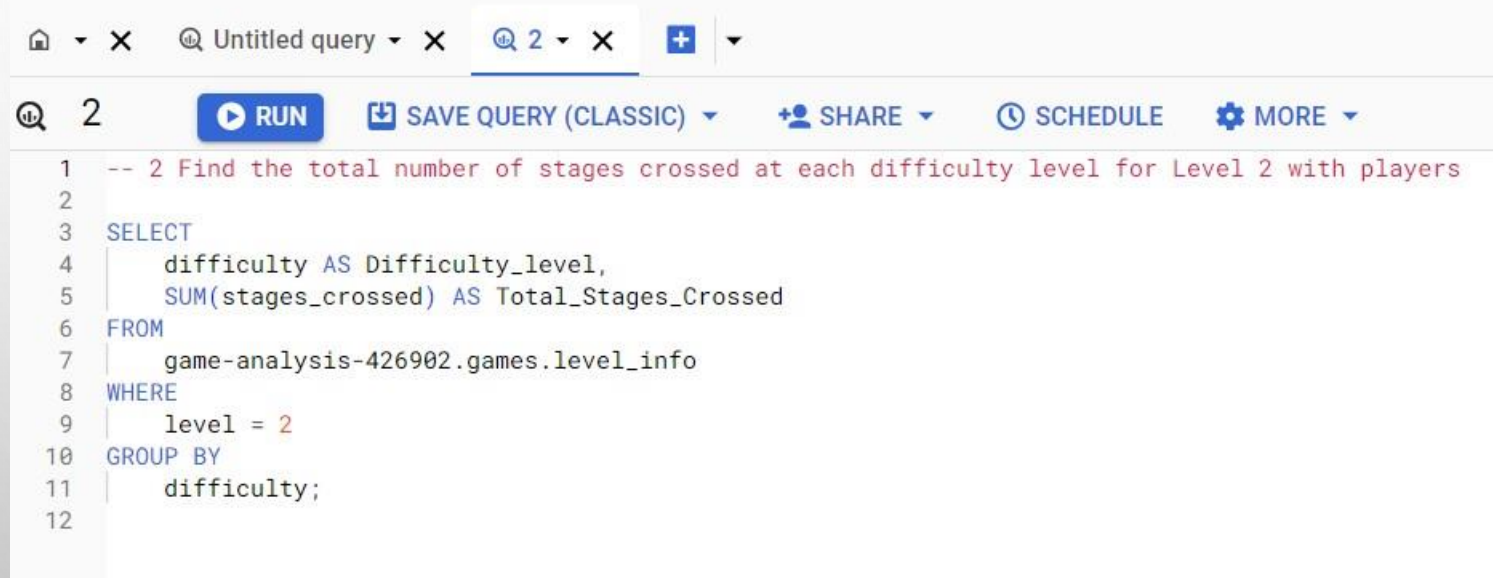
1. EXTRACT `P\_ID`, `DEV\_ID`, `PNAME`, AND `DIFFICULTY\_LEVEL` OF ALL PLAYERS AT LEVEL 0



The screenshot shows a SQL query editor interface. At the top, there is a toolbar with icons for home, close, search, and a dropdown menu showing 'Untitled query'. Below the toolbar, there is a search bar with '1' and a 'RUN' button. To the right of the 'RUN' button are buttons for 'SAVE QUERY (CLASSIC)', 'SHARE', 'SCHEDULE', and 'MORE'. The main area contains a SQL query with line numbers 1 through 13. The query is a SELECT statement that extracts player information from a database.

```
1  -- 1 Extract `P_ID`, `Dev_ID`, `PName`, and `Difficulty_level` of all players at Level 0
2
3  SELECT  li.P_ID,
4          li.Dev_ID,
5          pi.PName,
6          li.difficulty AS Difficulty_level
7
8  FROM    `game-analysis-426902.games.level_info` AS li
9
10 INNER JOIN game-analysis-426902.games.player_info AS pi
11 ON li.P_ID = pi.P_ID
12
13 WHERE li.Level = 0 ;
```

## 2. FIND THE TOTAL NUMBER OF STAGES CROSSED AT EACH DIFFICULTY LEVEL FOR LEVEL 2 WITH PLAYERS

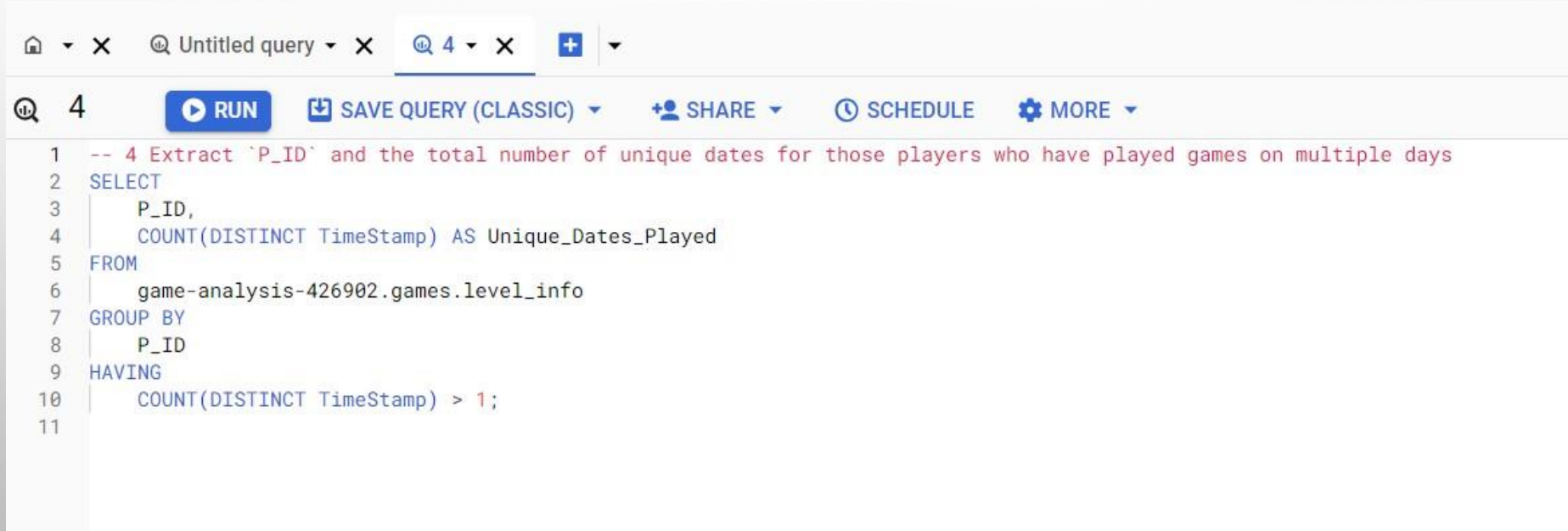


The screenshot shows a SQL query editor interface. At the top, there is a toolbar with icons for home, close, search, and a plus sign. Below the toolbar, there is a search bar with the text "2" and a "RUN" button. To the right of the "RUN" button are buttons for "SAVE QUERY (CLASSIC)", "SHARE", "SCHEDULE", and "MORE". The main area of the editor contains a SQL query with line numbers 1 through 12 on the left. The query is as follows:

```
1  -- 2 Find the total number of stages crossed at each difficulty level for Level 2 with players
2
3  SELECT
4      difficulty AS Difficulty_level,
5      SUM(stages_crossed) AS Total_Stages_Crossed
6  FROM
7      game-analysis-426902.games.level_info
8  WHERE
9      level = 2
10 GROUP BY
11     difficulty;
12
```



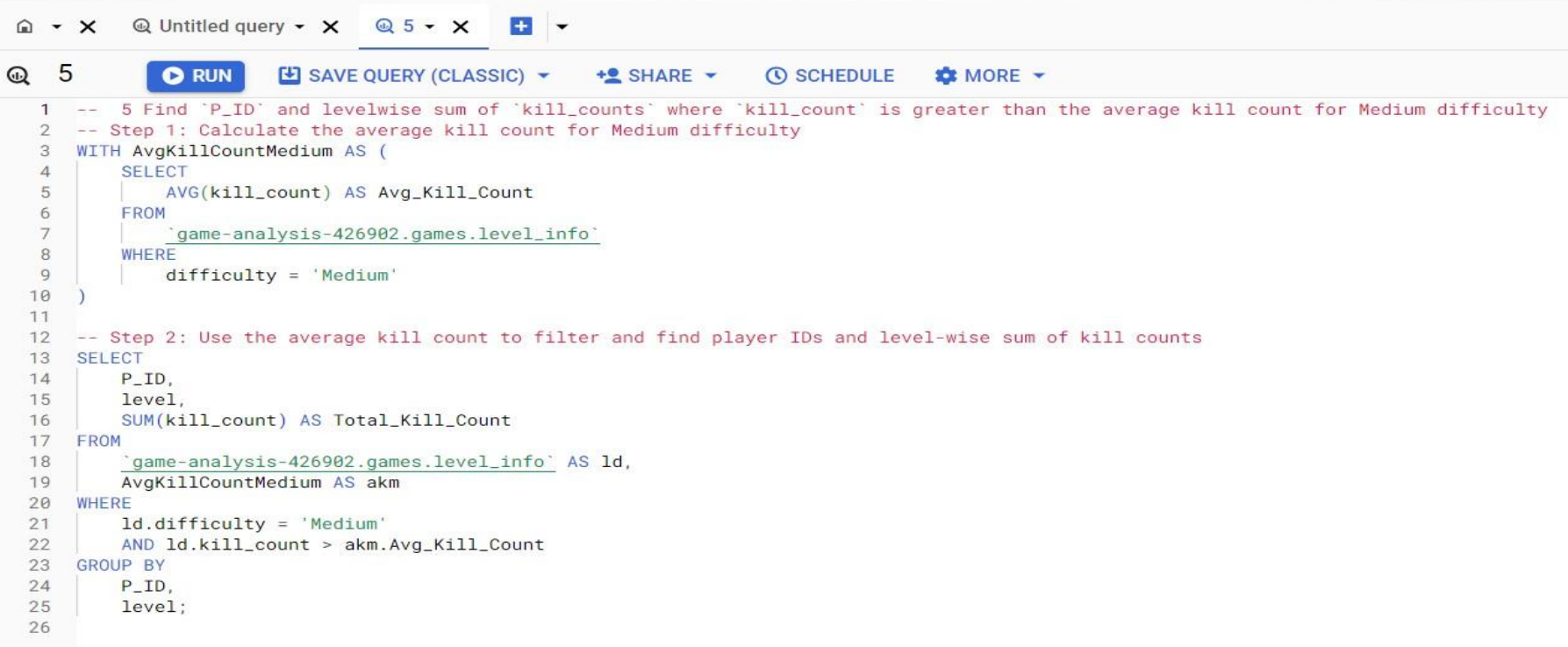
3. EXTRACT 'P\_ID' AND THE TOTAL NUMBER OF UNIQUE DATES FOR THOSE PLAYERS WHO HAVE PLAYED GAMES ON MULTIPLE DAYS



The screenshot shows a SQL query editor interface. At the top, there's a tab bar with 'Untitled query' and '4'. Below the tab bar is a toolbar with buttons for 'RUN', 'SAVE QUERY (CLASSIC)', 'SHARE', 'SCHEDULE', and 'MORE'. The main area contains a SQL query with line numbers 1 through 11. The query is a SELECT statement that filters for players who have played games on multiple days.

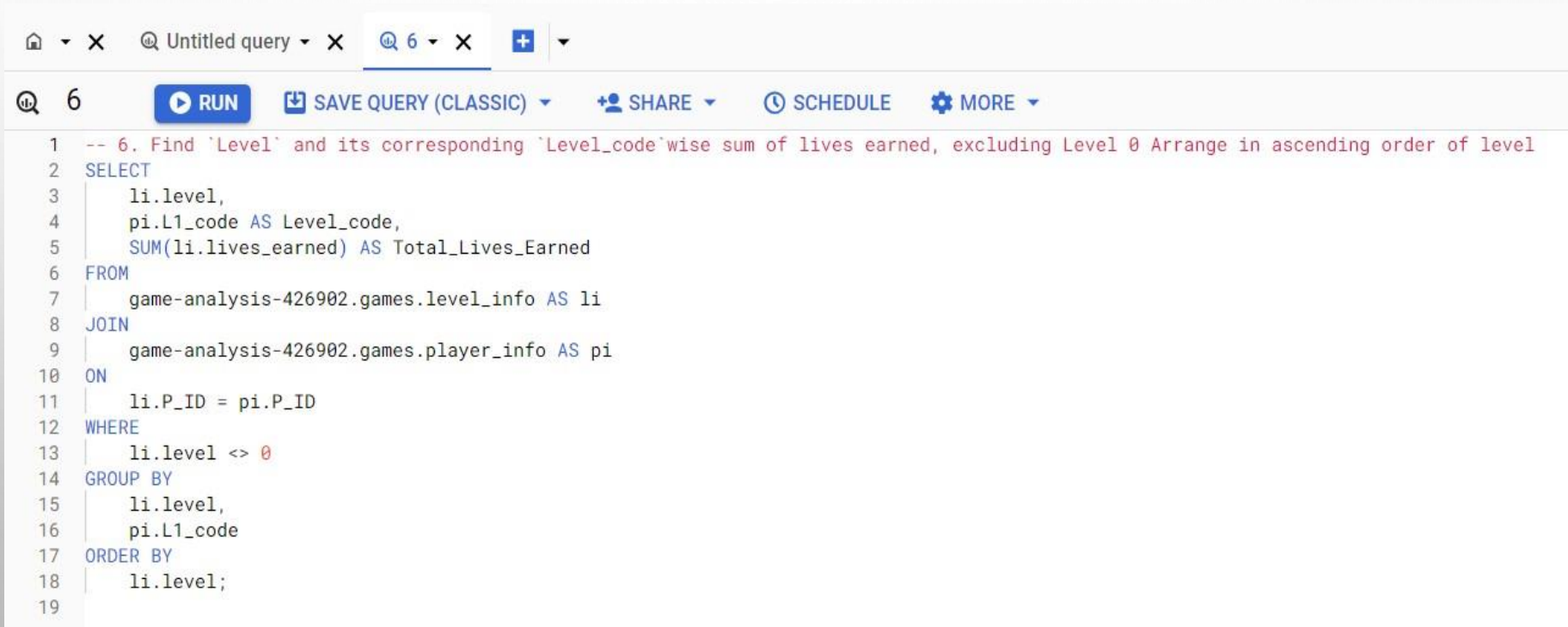
```
1 -- 4 Extract 'P_ID' and the total number of unique dates for those players who have played games on multiple days
2 SELECT
3     P_ID,
4     COUNT(DISTINCT TimeStamp) AS Unique_Dates_Played
5 FROM
6     game-analysis-426902.games.level_info
7 GROUP BY
8     P_ID
9 HAVING
10     COUNT(DISTINCT TimeStamp) > 1;
11
```

#### 4. FIND 'P\_ID' AND LEVELWISE SUM OF 'KILL\_COUNTS' WHERE 'KILL\_COUNT' IS GREATER THAN THE AVERAGE KILL COUNT FOR MEDIUM DIFFICULTY



```
1  -- 5 Find 'P_ID' and levelwise sum of 'kill_counts' where 'kill_count' is greater than the average kill count for Medium difficulty
2  -- Step 1: Calculate the average kill count for Medium difficulty
3  WITH AvgKillCountMedium AS (
4      SELECT
5          AVG(kill_count) AS Avg_Kill_Count
6      FROM
7          `game-analysis-426902.games.level_info`
8      WHERE
9          difficulty = 'Medium'
10 )
11
12 -- Step 2: Use the average kill count to filter and find player IDs and level-wise sum of kill counts
13 SELECT
14     P_ID,
15     level,
16     SUM(kill_count) AS Total_Kill_Count
17 FROM
18     `game-analysis-426902.games.level_info` AS ld,
19     AvgKillCountMedium AS akm
20 WHERE
21     ld.difficulty = 'Medium'
22     AND ld.kill_count > akm.Avg_Kill_Count
23 GROUP BY
24     P_ID,
25     level;
26
```

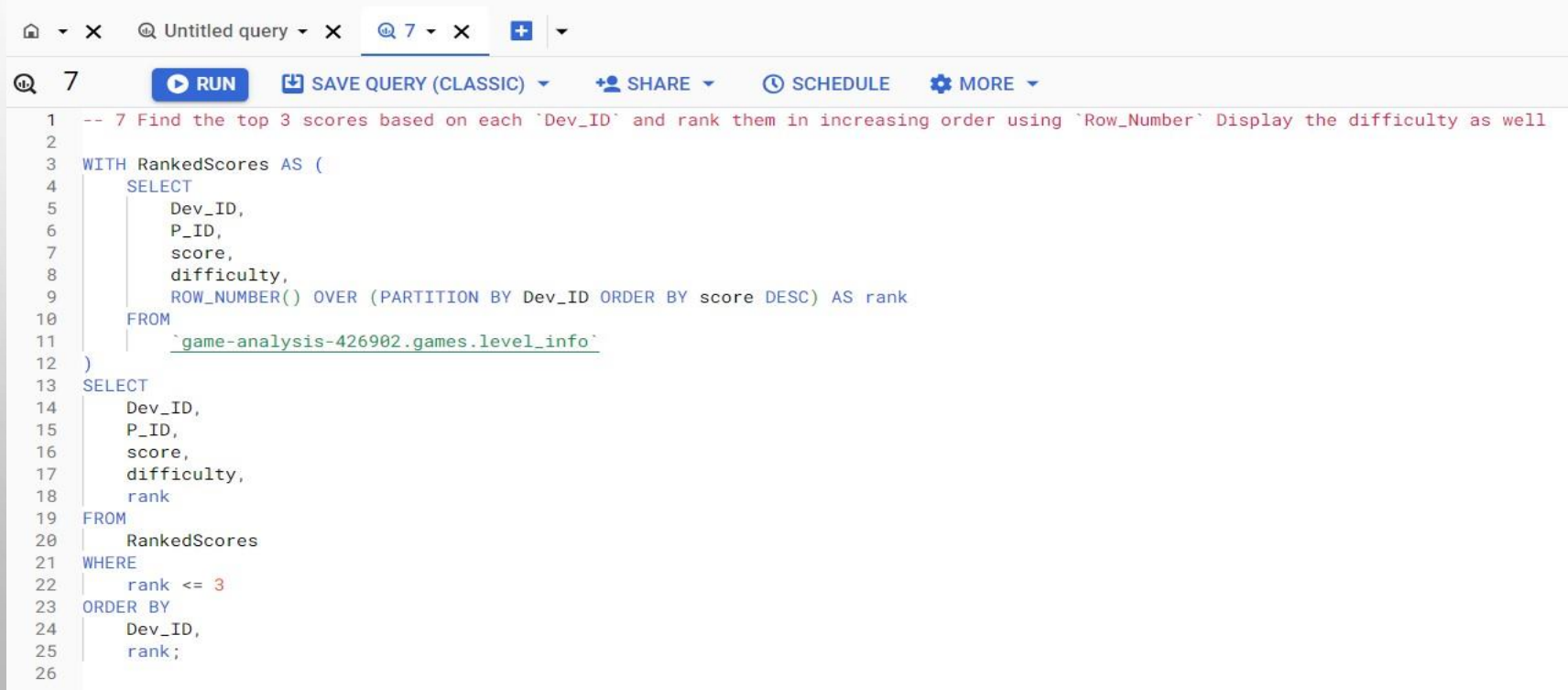
5. FIND 'LEVEL' AND ITS CORRESPONDING 'LEVEL\_CODE' WISE SUM OF LIVES EARNED, EXCLUDING LEVEL 0. ARRANGE IN ASCENDING ORDER OF LEVEL



The screenshot shows a SQL query editor interface. At the top, there's a tab labeled 'Untitled query' and a search bar with '6'. Below the tab, there's a toolbar with buttons for 'RUN', 'SAVE QUERY (CLASSIC)', 'SHARE', 'SCHEDULE', and 'MORE'. The main area contains a SQL query that is numbered 1 through 19. The query is as follows:

```
1  -- 6. Find 'Level' and its corresponding 'Level_code' wise sum of lives earned, excluding Level 0 Arrange in ascending order of level
2  SELECT
3      li.level,
4      pi.l1_code AS Level_code,
5      SUM(li.lives_earned) AS Total_Lives_Earned
6  FROM
7      game-analysis-426902.games.level_info AS li
8  JOIN
9      game-analysis-426902.games.player_info AS pi
10 ON
11     li.P_ID = pi.P_ID
12 WHERE
13     li.level <> 0
14 GROUP BY
15     li.level,
16     pi.l1_code
17 ORDER BY
18     li.level;
19
```

6. FIND THE TOP 3 SCORES BASED ON EACH `DEV\_ID` AND RANK THEM IN INCREASING ORDER USING `ROW\_NUMBER`. DISPLAY THE DIFFICULTY AS WELL



The screenshot shows a SQL query editor with a toolbar at the top containing icons for home, close, search, and tabs. The active tab is labeled '7'. The toolbar also includes buttons for 'RUN', 'SAVE QUERY (CLASSIC)', 'SHARE', 'SCHEDULE', and 'MORE'. The query text is as follows:

```
1  -- 7 Find the top 3 scores based on each `Dev_ID` and rank them in increasing order using `Row_Number` Display the difficulty as well
2
3  WITH RankedScores AS (
4      SELECT
5          Dev_ID,
6          P_ID,
7          score,
8          difficulty,
9          ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY score DESC) AS rank
10     FROM
11         `game-analysis-426902.games.level_info`
12 )
13 SELECT
14     Dev_ID,
15     P_ID,
16     score,
17     difficulty,
18     rank
19 FROM
20     RankedScores
21 WHERE
22     rank <= 3
23 ORDER BY
24     Dev_ID,
25     rank;
26
```

## 7. FIND THE 'FIRST\_LOGIN' DATETIME FOR EACH DEVICE ID



The screenshot shows a SQL query editor interface. At the top, there is a tab bar with a home icon, a close icon, a tab labeled 'Untitled query', and another tab labeled '8'. Below the tab bar is a toolbar with a search icon, a 'RUN' button, a 'SAVE QUERY (CLASSIC)' button, a 'SHARE' button, a 'SCHEDULE' button, and a 'MORE' button. The main area contains a SQL query with line numbers 1 through 8 on the left margin.

```
1 -- 8 Find the 'first_login' datetime for each device ID
2 SELECT
3     Dev_ID,
4     MIN(TimeStamp) AS first_login
5 FROM
6     `game-analysis-426902.games.level_info`
7 GROUP BY
8     Dev_ID;
```

8. FIND THE DEVICE ID THAT IS FIRST LOGGED IN (BASED ON `START\_DATETIME`) FOR EACH PLAYER (`P\_ID`). OUTPUT SHOULD CONTAIN PLAYER ID, DEVICE ID, AND FIRST LOGIN DATETIME.



The screenshot shows a SQL query editor interface. At the top, there's a toolbar with icons for home, close, search (magnifying glass), and a dropdown menu. Below the toolbar, there's a search bar with the text "10" and a "RUN" button. To the right of the "RUN" button are buttons for "SAVE QUERY (CLASSIC)", "SHARE", "SCHEDULE", and "MORE". On the far right, there's a green checkmark icon and the text "This que". The main area of the editor contains a SQL query with line numbers 1 through 21. The query is as follows:

```
1  /* 10 Find the device ID that is first logged in (based on `start_datetime`) for each player (`P_ID`) Output should contain player ID, device ID, and first login datetime */
2  WITH FirstLogin AS (
3      SELECT
4          P_ID,
5          Dev_ID,
6          TimeStamp,
7          ROW_NUMBER() OVER (PARTITION BY P_ID ORDER BY TimeStamp) AS rank
8      FROM
9          `game-analysis-426902.games.level_info`
10 )
11
12 SELECT
13     P_ID,
14     Dev_ID,
15     TimeStamp AS first_login
16 FROM
17     FirstLogin
18 WHERE
19     rank = 1
20 ORDER BY P_ID
21
```

## 9. EXTRACT THE TOP 3 HIGHEST SUMS OF SCORES FOR EACH `DEV\_ID` AND THE CORRESPONDING `P\_ID`

13

RUN

SAVE QUERY (CLASSIC)

SHARE

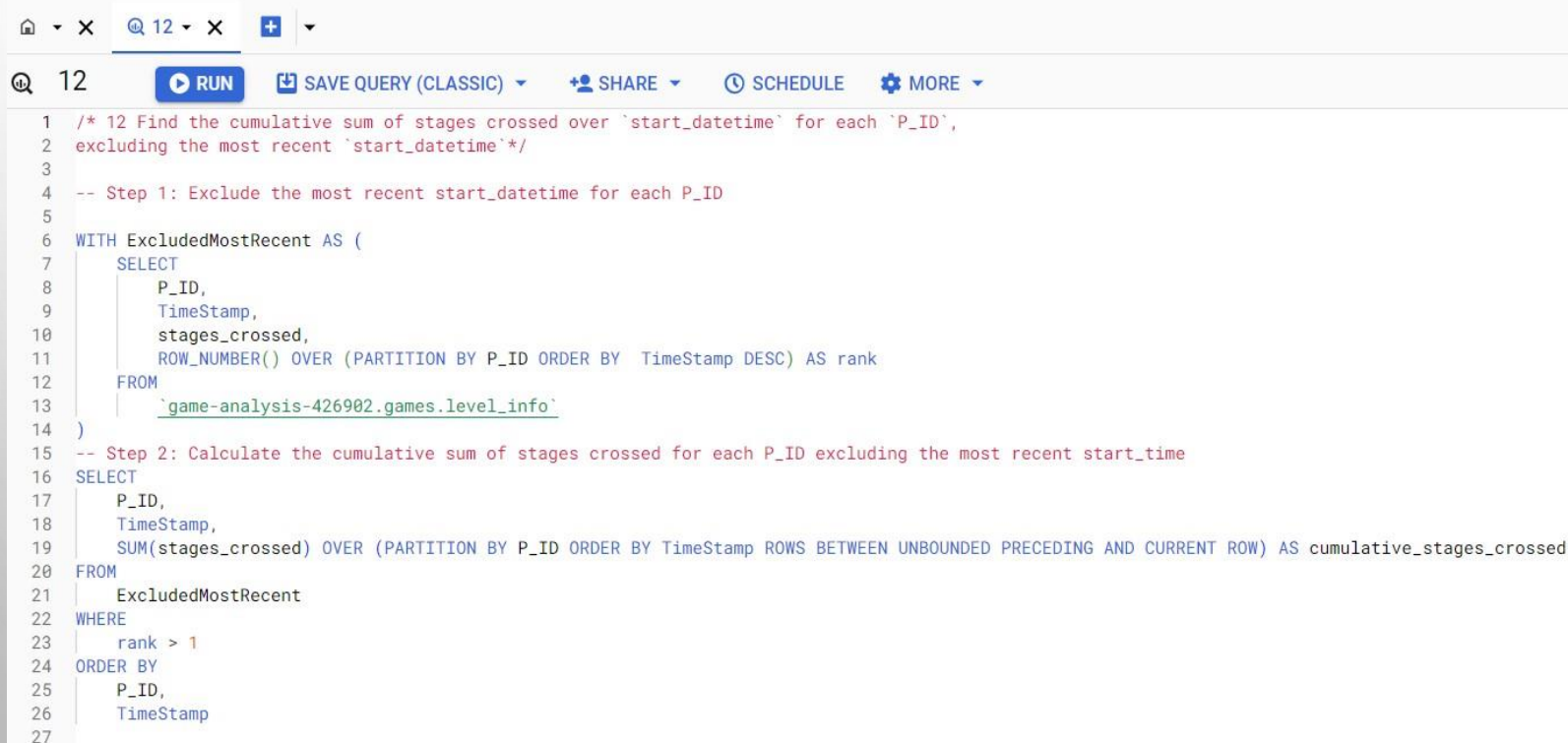
SCHEDULE

MORE

```
1  -- 13 Extract the top 3 highest sums of scores for each `Dev_ID` and the corresponding `P_ID`
2
3  WITH PlayerScores AS (
4      SELECT
5          Dev_ID,
6          P_ID,
7          SUM(score) AS Total_Score
8      FROM
9          `game-analysis-426902.games.level_info`
10     GROUP BY
11         Dev_ID,
12         P_ID
13 ),
14 RankedScores AS (
15     SELECT
16         Dev_ID,
17         P_ID,
18         Total_Score,
19         ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY Total_Score DESC) AS rank
20     FROM
21         PlayerScores
22 )
23 SELECT
24     Dev_ID,
25     P_ID,
26     Total_Score
27 FROM
28     RankedScores
29 WHERE
30     rank <= 3
31 ORDER BY
32     Dev_ID,
33     rank;
34
```



10. FIND THE CUMULATIVE SUM OF STAGES CROSSED OVER `START\_DATETIME` FOR EACH `P\_ID`, EXCLUDING THE MOST RECENT `START\_DATETIME`.



The screenshot shows a SQL query editor with a toolbar at the top containing icons for home, close, search, and a plus sign. Below the toolbar, the query is displayed with line numbers 1 through 27. The query is a SQL script that first excludes the most recent start\_datetime for each P\_ID and then calculates the cumulative sum of stages crossed for each P\_ID, excluding the most recent start\_time.

```
1  /* 12 Find the cumulative sum of stages crossed over `start_datetime` for each `P_ID`,
2  excluding the most recent `start_datetime` */
3
4  -- Step 1: Exclude the most recent start_datetime for each P_ID
5
6  WITH ExcludedMostRecent AS (
7      SELECT
8          P_ID,
9          TimeStamp,
10         stages_crossed,
11         ROW_NUMBER() OVER (PARTITION BY P_ID ORDER BY TimeStamp DESC) AS rank
12     FROM
13         `game-analysis-426902.games.level_info`
14 )
15 -- Step 2: Calculate the cumulative sum of stages crossed for each P_ID excluding the most recent start_time
16 SELECT
17     P_ID,
18     TimeStamp,
19     SUM(stages_crossed) OVER (PARTITION BY P_ID ORDER BY TimeStamp ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cumulative_stages_crossed
20 FROM
21     ExcludedMostRecent
22 WHERE
23     rank > 1
24 ORDER BY
25     P_ID,
26     TimeStamp
27
```



## KEY INSIGHTS AND RECOMMENDATIONS:

- Identified top-performing players and their performance metrics.
- Analyze player performance across different devices and difficulty levels.
- Discover trends and pattern in player engagement and behaviour.
- Suggested game experience improvements based on data driven insights.

**THANK YOU!**