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

QUESTIONS AND SQL QUERIES

1. EXTRACT `P_ID`, `DEV_ID`, `PNAME`, AND `DIFFICULTY_LEVEL` OF ALL PLAYERS AT LEVEL 0

Untitled query | 1

1 RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE

```
-- 1 Extract `P_ID`, `Dev_ID`, `PName`, and `Difficulty_level` of all players at Level 0

SELECT  li.P_ID,
        li.Dev_ID,
        pi.PName,
        li.difficulty AS Difficulty_level

FROM `game-analysis-426902.games.level_info` AS li

INNER JOIN game-analysis-426902.games.player_info AS pi
ON li.P_ID = pi.P_ID

WHERE li.Level = 0 ;
```

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

Untitled query | 2

2 RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE

```sql
-- 2 Find the total number of stages crossed at each difficulty level for Level 2 with players

SELECT
    difficulty AS Difficulty_level,
    SUM(stages_crossed) AS Total_Stages_Crossed
FROM
    game-analysis-426902.games.level_info
WHERE
    level = 2
GROUP BY
    difficulty;
```

3. EXTRACT `P_ID` AND THE TOTAL NUMBER OF UNIQUE DATES FOR THOSE PLAYERS WHO HAVE PLAYED GAMES ON MULTIPLE DAYS

Untitled query | 4

4 RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE

```sql
-- 4 Extract `P_ID` and the total number of unique dates for those players who have played games on multiple days
SELECT
    P_ID,
    COUNT(DISTINCT TimeStamp) AS Unique_Dates_Played
FROM
    game-analysis-426902.games.level_info
GROUP BY
    P_ID
HAVING
    COUNT(DISTINCT TimeStamp) > 1;
```

4. FIND `P_ID` AND LEVELWISE SUM OF `KILL_COUNTS` WHERE `KILL_COUNT` IS GREATER THAN THE AVERAGE KILL COUNT FOR MEDIUM DIFFICULTY

Untitled query | 5

5 RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE

```sql
--  5 Find `P_ID` and levelwise sum of `kill_counts` where `kill_count` is greater than the average kill count for Medium difficulty
-- Step 1: Calculate the average kill count for Medium difficulty
WITH AvgKillCountMedium AS (
    SELECT
        AVG(kill_count) AS Avg_Kill_Count
    FROM
        `game-analysis-426902.games.level_info`
    WHERE
        difficulty = 'Medium'
)

-- Step 2: Use the average kill count to filter and find player IDs and level-wise sum of kill counts
SELECT
    P_ID,
    level,
    SUM(kill_count) AS Total_Kill_Count
FROM
    `game-analysis-426902.games.level_info` AS ld,
    AvgKillCountMedium AS akm
WHERE
    ld.difficulty = 'Medium'
    AND ld.kill_count > akm.Avg_Kill_Count
GROUP BY
    P_ID,
    level;
```

5. FIND `LEVEL` AND ITS CORRESPONDING `LEVEL_CODE`WISE SUM OF LIVES EARNED, EXCLUDING LEVEL 0. ARRANGE IN ASCENDING ORDER OF LEVEL

Untitled query | 6

6 RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE

```
-- 6. Find `Level` and its corresponding `Level_code`wise sum of lives earned, excluding Level 0 Arrange in ascending order of level
SELECT
    li.level,
    pi.L1_code AS Level_code,
    SUM(li.lives_earned) AS Total_Lives_Earned
FROM
    game-analysis-426902.games.level_info AS li
JOIN
    game-analysis-426902.games.player_info AS pi
ON
    li.P_ID = pi.P_ID
WHERE
    li.level <> 0
GROUP BY
    li.level,
    pi.L1_code
ORDER BY
    li.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

Untitled query | 7

7 RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE

```
-- 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

WITH RankedScores AS (
    SELECT
        Dev_ID,
        P_ID,
        score,
        difficulty,
        ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY score DESC) AS rank
    FROM
        `game-analysis-426902.games.level_info`
)
SELECT
    Dev_ID,
    P_ID,
    score,
    difficulty,
    rank
FROM
    RankedScores
WHERE
    rank <= 3
ORDER BY
    Dev_ID,
    rank;
```

7. FIND THE `FIRST_LOGIN` DATETIME FOR EACH DEVICE ID

Untitled query | 8

8 RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE

```sql
-- 8 Find the `first_login` datetime for each device ID
SELECT
    Dev_ID,
    MIN(TimeStamp) AS first_login
FROM
    `game-analysis-426902.games.level_info`
GROUP BY
    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.

10 RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE This que

```
/* 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 */
WITH FirstLogin AS (
    SELECT
        P_ID,
        Dev_ID,
        TimeStamp,
        ROW_NUMBER() OVER (PARTITION BY P_ID ORDER BY TimeStamp) AS rank
    FROM
        `game-analysis-426902.games.level_info`
)

SELECT
    P_ID,
    Dev_ID,
    TimeStamp AS first_login
FROM
    FirstLogin
WHERE
    rank = 1
ORDER BY P_ID
```

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

```sql
-- 13 Extract the top 3 highest sums of scores for each `Dev_ID` and the corresponding `P_ID`

WITH PlayerScores AS (
    SELECT
        Dev_ID,
        P_ID,
        SUM(score) AS Total_Score
    FROM
        `game-analysis-426902.games.level_info`
    GROUP BY
        Dev_ID,
        P_ID
),
RankedScores AS (
    SELECT
        Dev_ID,
        P_ID,
        Total_Score,
        ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY Total_Score DESC) AS rank
    FROM
        PlayerScores
)
SELECT
    Dev_ID,
    P_ID,
    Total_Score
FROM
    RankedScores
WHERE
    rank <= 3
ORDER BY
    Dev_ID,
    rank;
```

10. FIND THE CUMULATIVE SUM OF STAGES CROSSED OVER `START_DATETIME` FOR EACH `P_ID`, EXCLUDING THE MOST RECENT `START_DATETIME`.

12

RUN SAVE QUERY (CLASSIC) SHARE SCHEDULE MORE

```
/* 12 Find the cumulative sum of stages crossed over `start_datetime` for each `P_ID`,
excluding the most recent `start_datetime`*/

-- Step 1: Exclude the most recent start_datetime for each P_ID

WITH ExcludedMostRecent AS (
    SELECT
        P_ID,
        TimeStamp,
        stages_crossed,
        ROW_NUMBER() OVER (PARTITION BY P_ID ORDER BY  TimeStamp DESC) AS rank
    FROM
        `game-analysis-426902.games.level_info`
)
-- Step 2: Calculate the cumulative sum of stages crossed for each P_ID excluding the most recent start_time
SELECT
    P_ID,
    TimeStamp,
    SUM(stages_crossed) OVER (PARTITION BY P_ID ORDER BY TimeStamp ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cumulative_stages_crossed
FROM
    ExcludedMostRecent
WHERE
    rank > 1
ORDER BY
    P_ID,
    TimeStamp
```

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!