DECODE GAMING BEHAVIOUR WITH SQL

MENTORNESS INTERNSHIP PROJECT BY

VICTOR SIDI

BATCH – MIP-DA-06

TODAY'S AGENDA

Project overview and objective

Data Cleaning

Data Analysis

Insights and Recommendations

Summary

Project Overview and Objective

Project Overview: The "Decode Gaming Behavior" project focuses on analyzing player behavior, level progression, and gameplay dynamics using SQL queries on a game dataset comprising two tables: Player Details and Level Details.

Objective: The primary goal of this project is to extract valuable insights from the game dataset to understand player behavior and game dynamics. This includes analyzing various aspects such as player progression, performance metrics, and device interactions.

DATASET DESCRIPTION

Player Details Table:

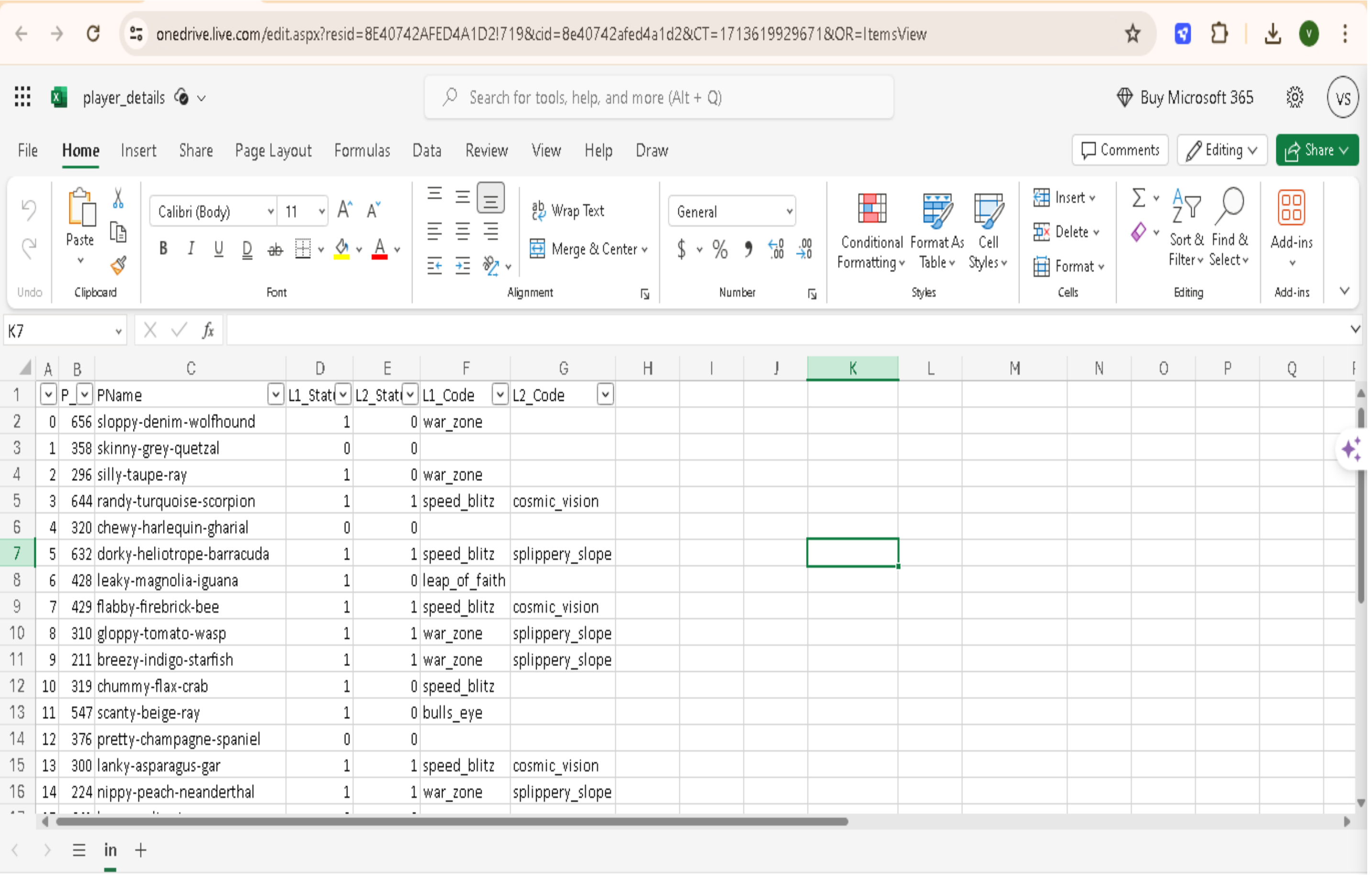
- `P_ID`: Player ID
- `PName`: Player Name
- `L1_status`: Level 1 Status
- `L2_status`: Level 2 Status
- `L1_code`: System-generated Level 1 Code
- `L2_code`: System-generated Level 2 Code

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

DATA BEFORE PRE-PROCESSING

DATA CLEANING



	A	B	C	D	E	F	G
1		P_	PName	L1_Stat	L2_Stat	L1_Code	L2_Code
2	0	656	sloppy-denim-wolfhound	1	0	war_zone	
3	1	358	skinny-grey-quetzal	0	0		
4	2	296	silly-taupe-ray	1	0	war_zone	
5	3	644	randy-turquoise-scorpion	1	1	speed_blitz	cosmic_vision
6	4	320	chewy-harlequin-gharial	0	0		
7	5	632	dorky-heliotrope-barracuda	1	1	speed_blitz	splippery_slope
8	6	428	leaky-magnolia-iguana	1	0	leap_of_faith	
9	7	429	flabby-firebrick-bee	1	1	speed_blitz	cosmic_vision
10	8	310	gloppy-tomato-wasp	1	1	war_zone	splippery_slope
11	9	211	breezy-indigo-starfish	1	1	war_zone	splippery_slope
12	10	319	chummy-flax-crab	1	0	speed_blitz	
13	11	547	scanty-beige-ray	1	0	bulls_eye	
14	12	376	pretty-champagne-spaniel	0	0		
15	13	300	lanky-asparagus-gar	1	1	speed_blitz	cosmic_vision
16	14	224	nippy-peach-neanderthal	1	1	war_zone	splippery_slope

DATA IMPUTATION:

REPLACE THE MISSING VALUES WITH SUITABLE SUBSTITUTES SUCH AS MEAN, MEDIAN, OR MODE.

CALCULATE THE MODE FOR VALUES IN THE COLUMNS L1_CODE AND L2_CODE

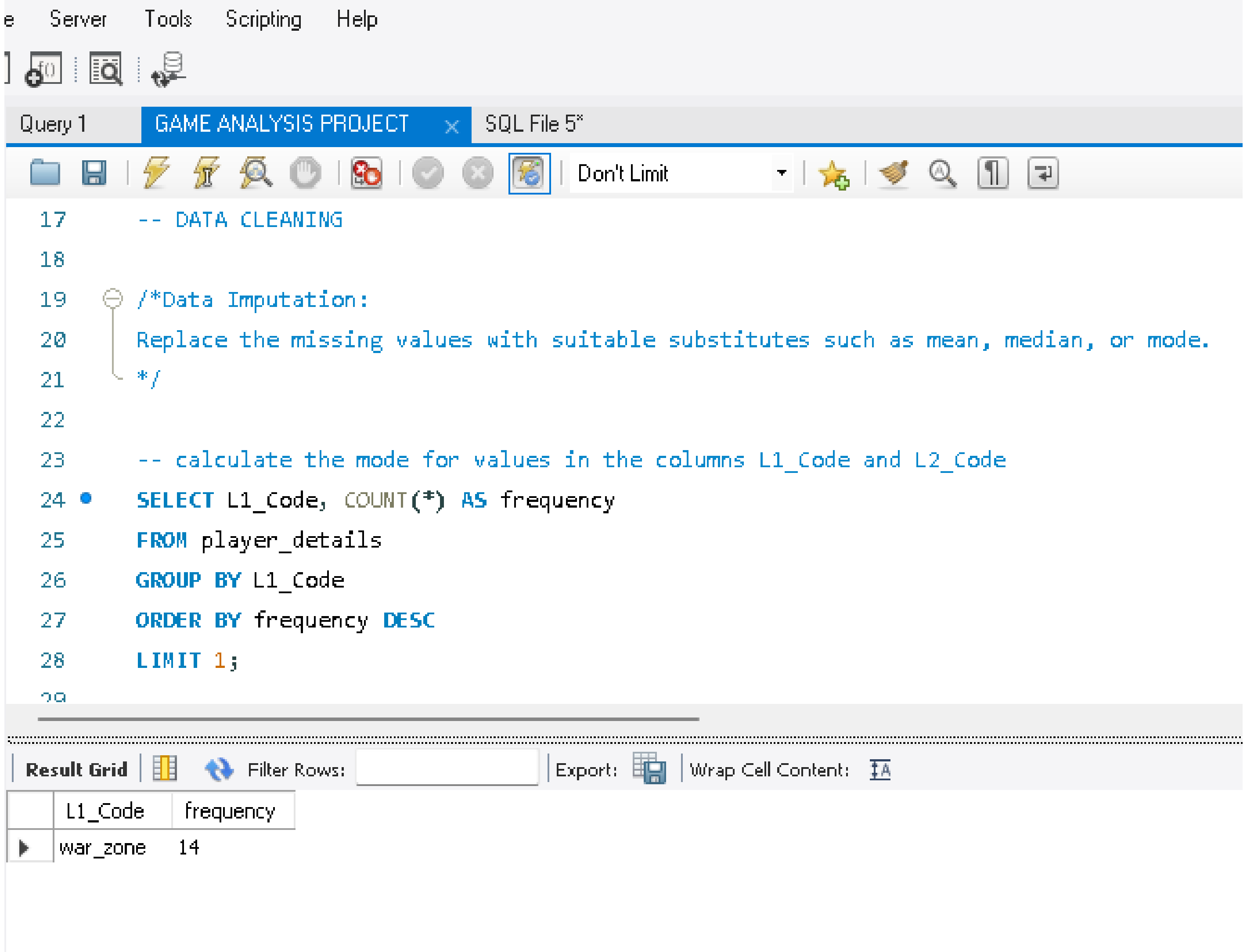
SELECT L1_CODE, COUNT(*) AS FREQUENCY

FROM PLAYER_DETAILS

GROUP BY L1_CODE

ORDER BY FREQUENCY DESC

LIMIT 1;



```sql
-- DATA CLEANING

/*Data Imputation:
Replace the missing values with suitable substitutes such as mean, median, or mode.
*/

-- calculate the mode for values in the columns L1_Code and L2_Code
SELECT L1_Code, COUNT(*) AS frequency
FROM player_details
GROUP BY L1_Code
ORDER BY frequency DESC
LIMIT 1;
```

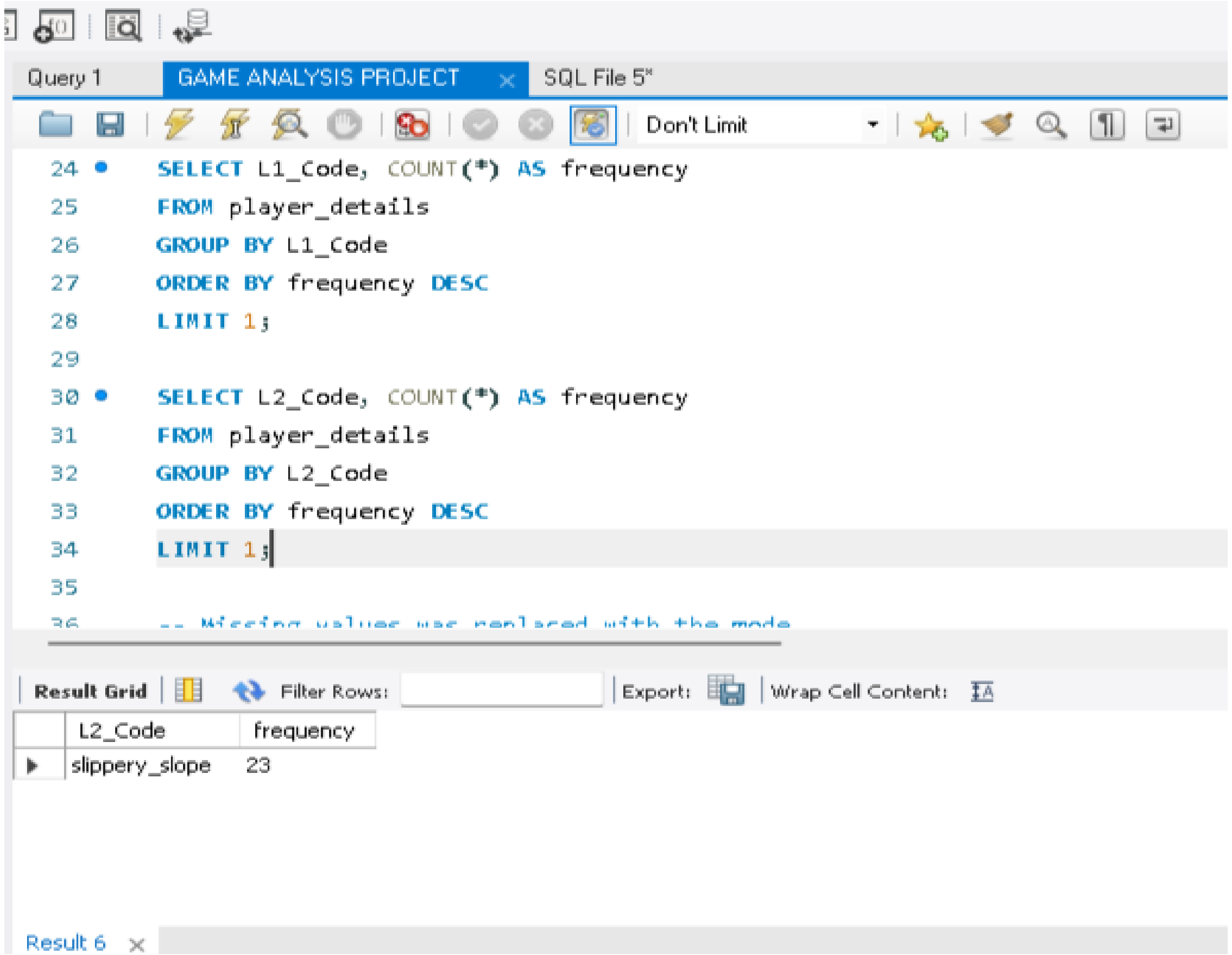
L1_Code	frequency
war_zone	14

DATA IMPUTATION:

REPLACE THE MISSING VALUES WITH SUITABLE SUBSTITUTES SUCH AS MEAN, MEDIAN, OR MODE.

CALCULATE THE MODE FOR VALUES IN THE COLUMNS L1_CODE AND L2_CODE

```
SELECT L2_CODE, COUNT(*)
AS FREQUENCY

FROM PLAYER_DETAILS

GROUP BY L2_CODE

ORDER BY FREQUENCY DESC

LIMIT 1;
```

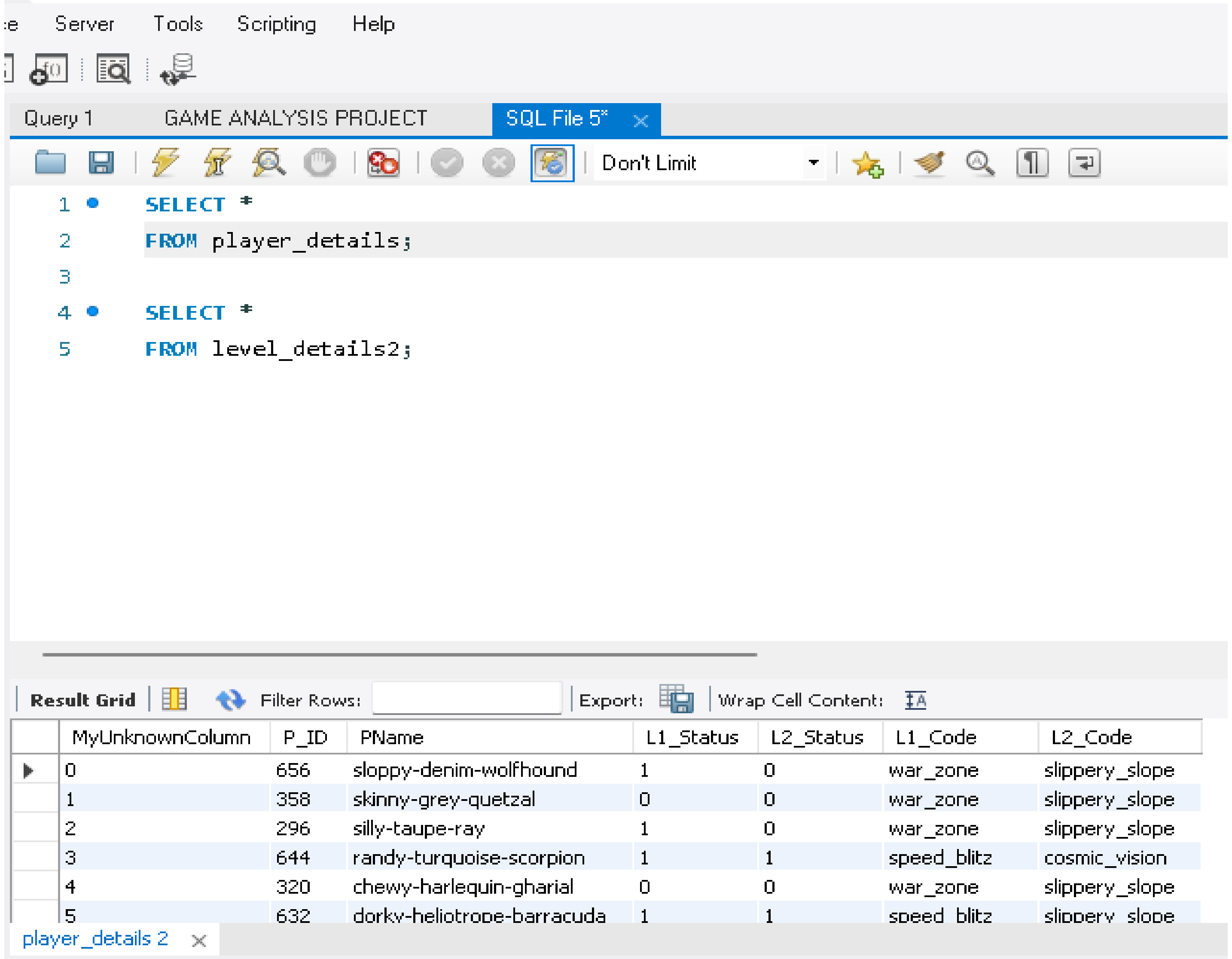


```
SELECT L1_Code, COUNT(*) AS frequency
FROM player_details
GROUP BY L1_Code
ORDER BY frequency DESC
LIMIT 1;

SELECT L2_Code, COUNT(*) AS frequency
FROM player_details
GROUP BY L2_Code
ORDER BY frequency DESC
LIMIT 1;

-- Missing values was replaced with the mode
```

L2_Code	frequency
slippery_slope	23

DATA AFTER PRE-PROCESSING

DATA CLEANING



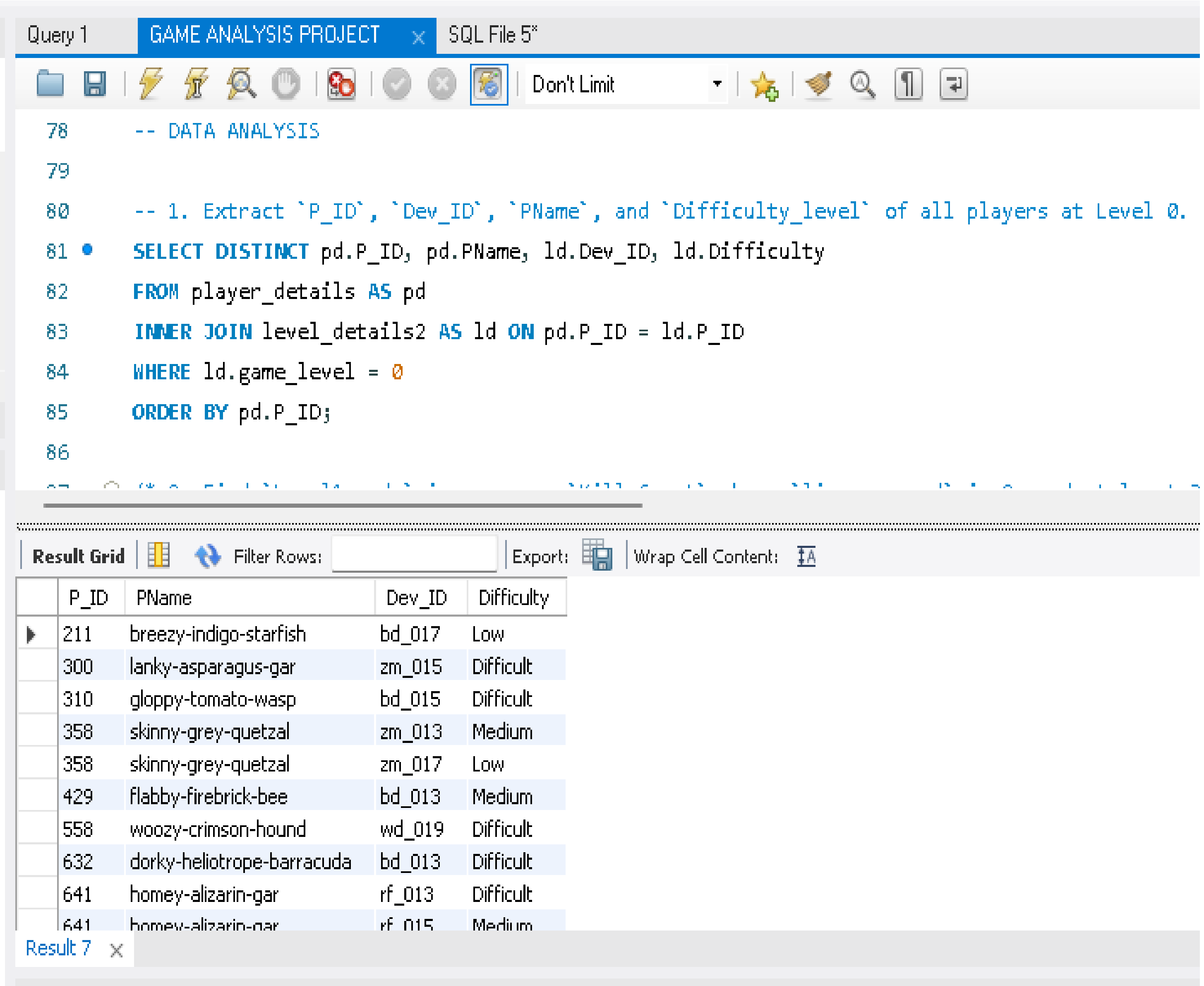
```
SELECT *
FROM player_details;

SELECT *
FROM level_details2;
```

MyUnknownColumn	P_ID	PName	L1_Status	L2_Status	L1_Code	L2_Code
0	656	sloppy-denim-wolfhound	1	0	war_zone	slippery_slope
1	358	skinny-grey-quetzal	0	0	war_zone	slippery_slope
2	296	silly-taupe-ray	1	0	war_zone	slippery_slope
3	644	randy-turquoise-scorpion	1	1	speed_blitz	cosmic_vision
4	320	chewy-harlequin-gharial	0	0	war_zone	slippery_slope
5	632	dorky-heliotrope-barracuda	1	1	speed_blitz	slippery_slope

DATA ANALYSIS

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

```
SELECT DISTINCT PD.P_ID, PD.PNAME, LD.DEV_ID, LD.DIFFICULTY

FROM PLAYER_DETAILS AS PD

INNER JOIN LEVEL_DETAILS2 AS LD ON PD.P_ID = LD.P_ID

WHERE LD.GAME_LEVEL = 0

ORDER BY PD.P_ID;
```



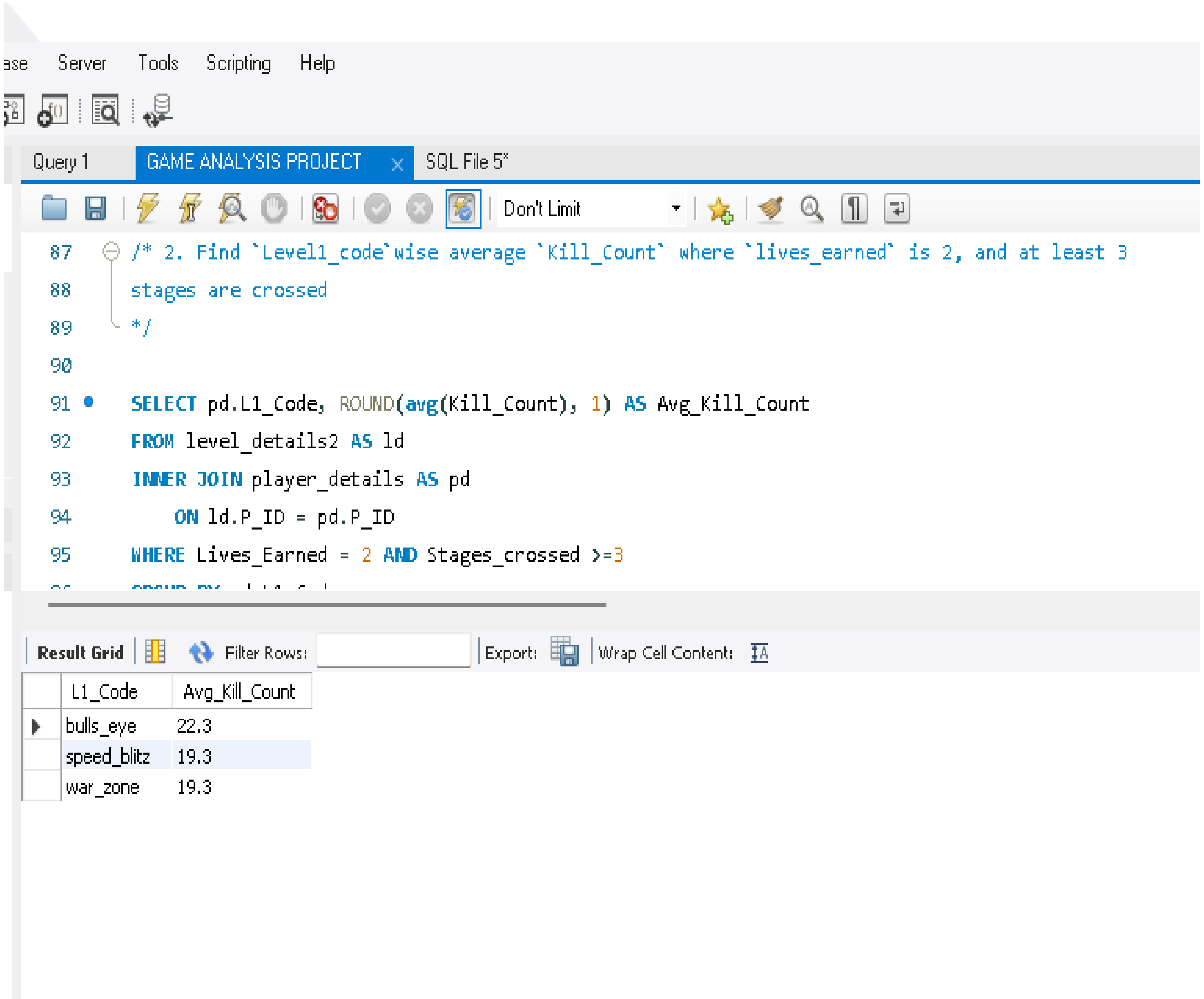
```
-- DATA ANALYSIS

-- 1. Extract `P_ID`, `Dev_ID`, `PName`, and `Difficulty_level` of all players at Level 0.
SELECT DISTINCT pd.P_ID, pd.PName, ld.Dev_ID, ld.Difficulty
FROM player_details AS pd
INNER JOIN level_details2 AS ld ON pd.P_ID = ld.P_ID
WHERE ld.game_level = 0
ORDER BY pd.P_ID;

```

P_ID	PName	Dev_ID	Difficulty
211	breezy-indigo-starfish	bd_017	Low
300	lanky-asparagus-gar	zm_015	Difficult
310	gloppy-tomato-wasp	bd_015	Difficult
358	skinny-grey-quetzal	zm_013	Medium
358	skinny-grey-quetzal	zm_017	Low
429	flabby-firebrick-bee	bd_013	Medium
558	woozy-crimson-hound	wd_019	Difficult
632	dorky-heliotrope-barracuda	bd_013	Difficult
641	homey-alizarin-gar	rf_013	Difficult
641	homey-alizarin-gar	rf_015	Medium

DATA ANALYSIS

2. FIND `LEVEL1_CODE`WISE AVERAGE `KILL_COUNT` WHERE `LIVES_EARNED` IS 2, AND AT LEAST 3 STAGES ARE CROSSED

```
SELECT PD.L1_CODE, ROUND(AVG(KILL_COUNT), 1) AS AVG_KILL_COUNT

FROM LEVEL_DETAILS2 AS LD

INNER JOIN PLAYER_DETAILS AS PD

    ON LD.P_ID = PD.P_ID

WHERE LIVES_EARNED = 2 AND STAGES_CROSSED >=3

GROUP BY PD.L1_CODE

ORDER BY AVG_KILL_COUNT DESC;
```



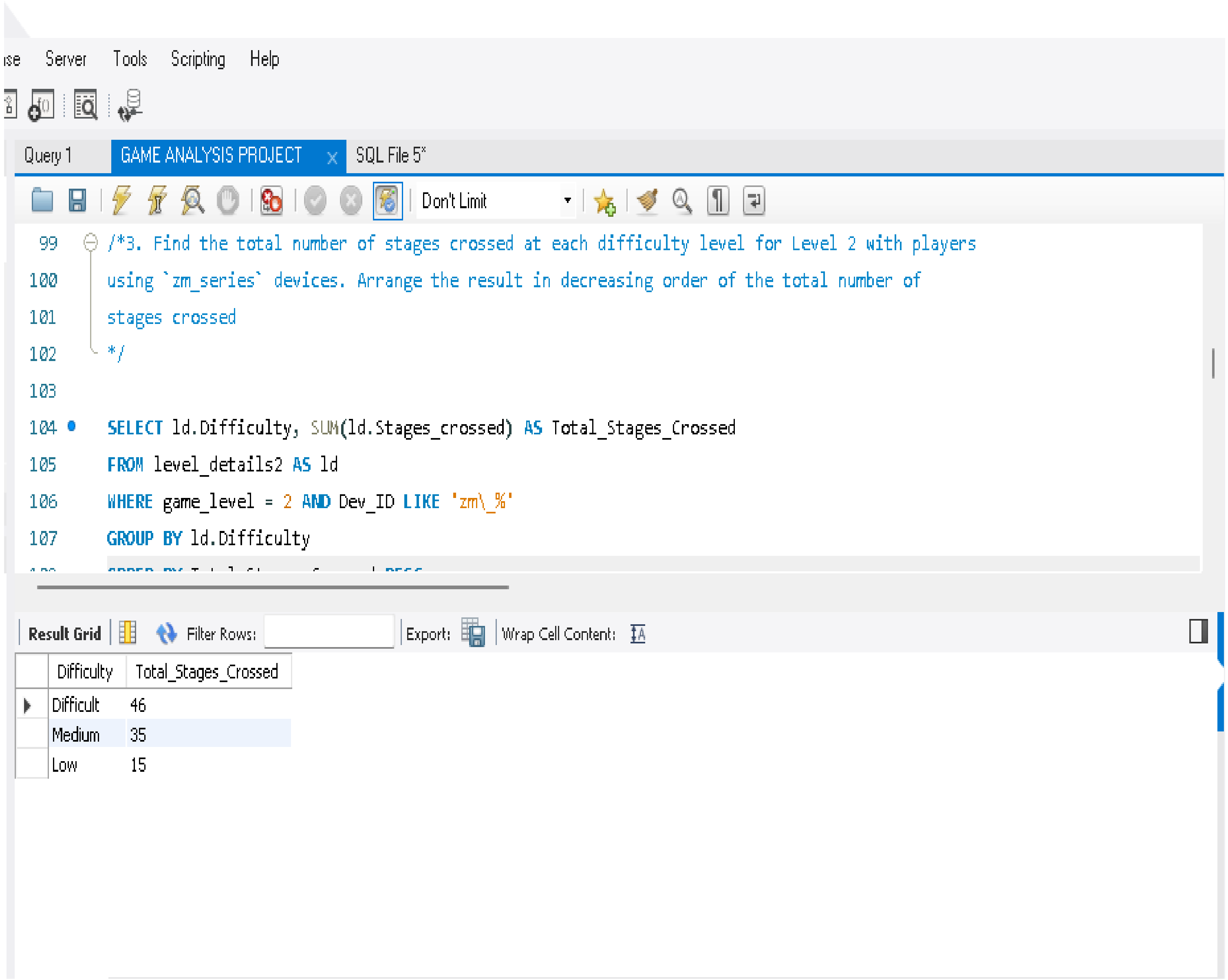
```
/* 2. Find `Level1_code`wise average `Kill_Count` where `lives_earned` is 2, and at least 3
stages are crossed
*/

SELECT pd.L1_Code, ROUND(avg(Kill_Count), 1) AS Avg_Kill_Count
FROM level_details2 AS ld
INNER JOIN player_details AS pd
    ON ld.P_ID = pd.P_ID
WHERE Lives_Earned = 2 AND Stages_crossed >=3
```

L1_Code	Avg_Kill_Count
bulls_eye	22.3
speed_blitz	19.3
war_zone	19.3

DATA ANALYSIS

3. FIND THE TOTAL NUMBER OF STAGES CROSSED AT EACH DIFFICULTY LEVEL FOR LEVEL 2 WITH PLAYERS USING `ZM_SERIES` DEVICES. ARRANGE THE RESULT IN DECREASING ORDER OF THE TOTAL NUMBER OF STAGES CROSSED

```
SELECT LD.DIFFICULTY,
SUM(LD.STAGES_CROSSED) AS
TOTAL_STAGES_CROSSED

FROM LEVEL_DETAILS2 AS LD

WHERE GAME_LEVEL = 2 AND
DEV_ID LIKE 'ZM\_%'

GROUP BY LD.DIFFICULTY

ORDER BY TOTAL_STAGES_CROSSED
DESC;
```



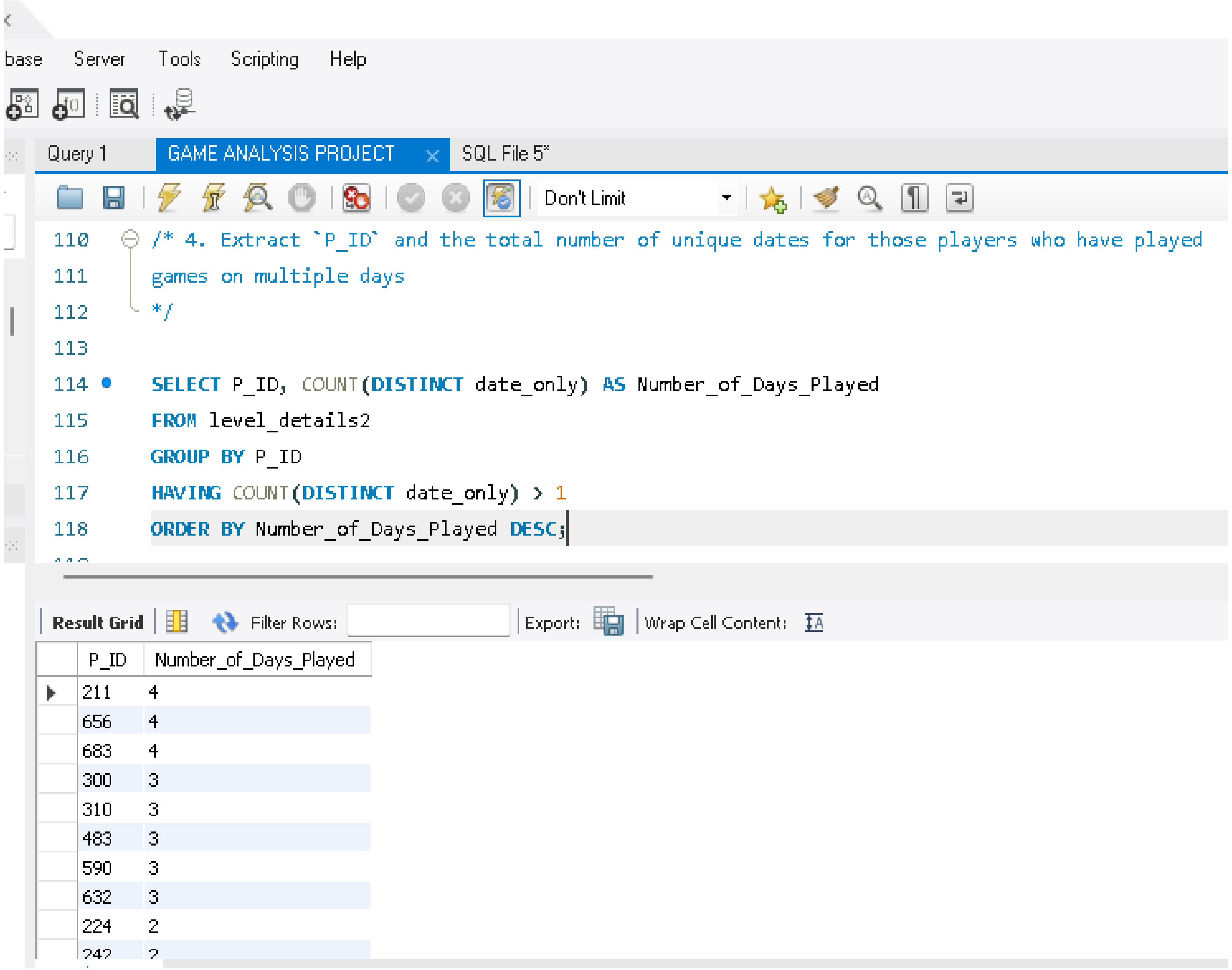
```
/*3. Find the total number of stages crossed at each difficulty level for Level 2 with players
using `zm_series` devices. Arrange the result in decreasing order of the total number of
stages crossed
*/

SELECT ld.Difficulty, SUM(ld.Stages_crossed) AS Total_Stages_Crossed
FROM level_details2 AS ld
WHERE game_level = 2 AND Dev_ID LIKE 'zm\_%'
GROUP BY ld.Difficulty
ORDER BY Total_Stages_Crossed DESC
```

Difficulty	Total_Stages_Crossed
Difficult	46
Medium	35
Low	15

DATA ANALYSIS

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
DATE_ONLY) AS
NUMBER_OF_DAYS_PLAYED

FROM LEVEL_DETAILS2

GROUP BY P_ID

HAVING COUNT(DISTINCT
DATE_ONLY) > 1

ORDER BY NUMBER_OF_DAYS_PLAYED
DESC;
```



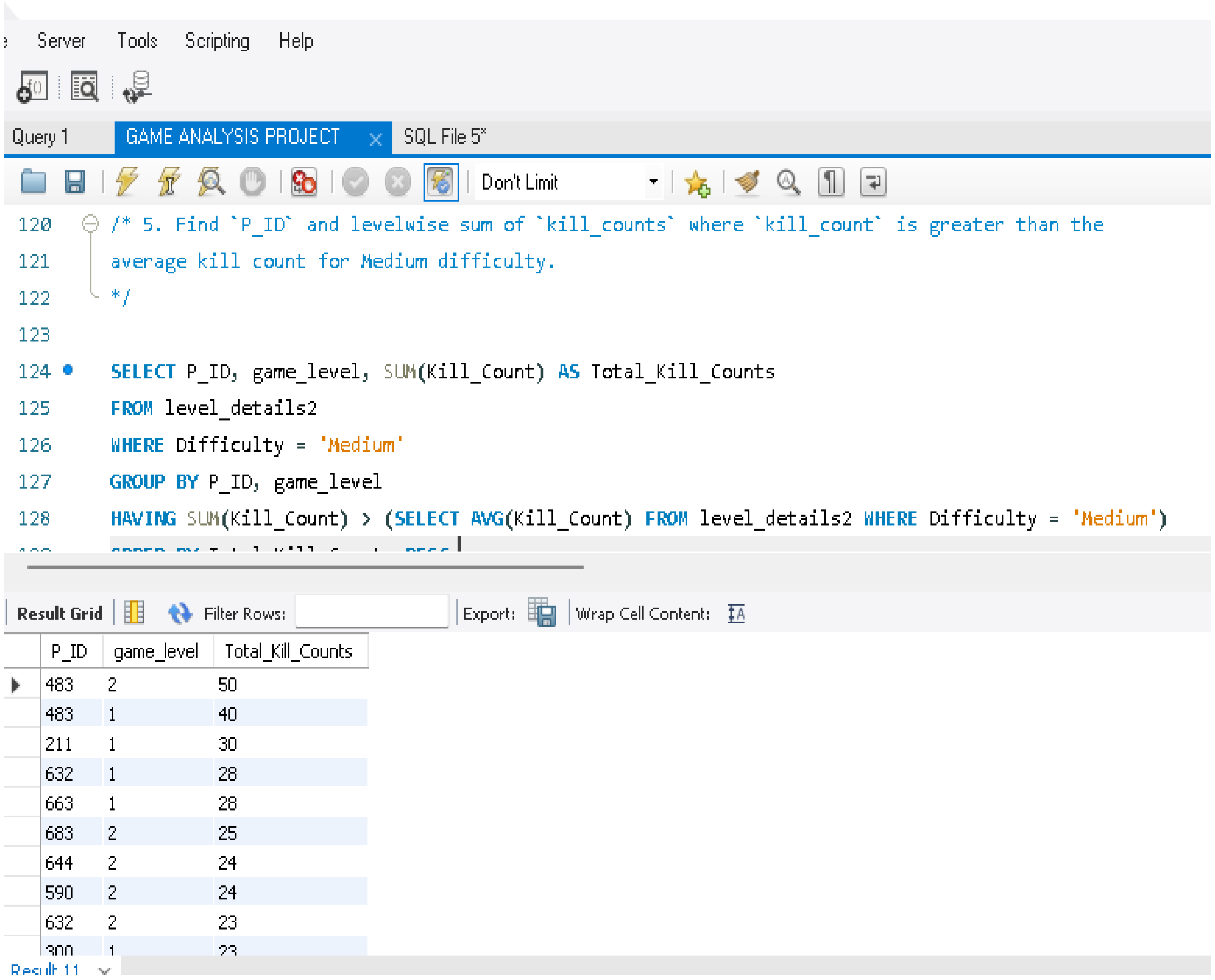
```
/* 4. Extract `P_ID` and the total number of unique dates for those players who have played
games on multiple days
*/

SELECT P_ID, COUNT(DISTINCT date_only) AS Number_of_Days_Played
FROM level_details2
GROUP BY P_ID
HAVING COUNT(DISTINCT date_only) > 1
ORDER BY Number_of_Days_Played DESC;
```

P_ID	Number_of_Days_Played
211	4
656	4
683	4
300	3
310	3
483	3
590	3
632	3
224	2
242	2

DATA ANALYSIS

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

```
SELECT P_ID, GAME_LEVEL,
SUM(KILL_COUNT) AS
TOTAL_KILL_COUNTS

FROM LEVEL_DETAILS2

WHERE DIFFICULTY = 'MEDIUM'

GROUP BY P_ID, GAME_LEVEL

HAVING SUM(KILL_COUNT) >
(SELECT AVG(KILL_COUNT) FROM
LEVEL_DETAILS2 WHERE
DIFFICULTY = 'MEDIUM')

ORDER BY TOTAL_KILL_COUNTS
DESC;
```



```
/* 5. Find `P_ID` and levelwise sum of `kill_counts` where `kill_count` is greater than the
average kill count for Medium difficulty.
*/

SELECT P_ID, game_level, SUM(Kill_Count) AS Total_Kill_Counts
FROM level_details2
WHERE Difficulty = 'Medium'
GROUP BY P_ID, game_level
HAVING SUM(Kill_Count) > (SELECT AVG(Kill_Count) FROM level_details2 WHERE Difficulty = 'Medium')
```

P_ID	game_level	Total_Kill_Counts
483	2	50
483	1	40
211	1	30
632	1	28
663	1	28
683	2	25
644	2	24
590	2	24
632	2	23
300	1	23

DATA ANALYSIS

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

SELECT LD.GAME_LEVEL, PLAYER_DETAILS.L1_CODE, PLAYER_DETAILS.L2_CODE, SUM(LD.LIVES_EARNED) AS TOTAL_LIVES_EARNED

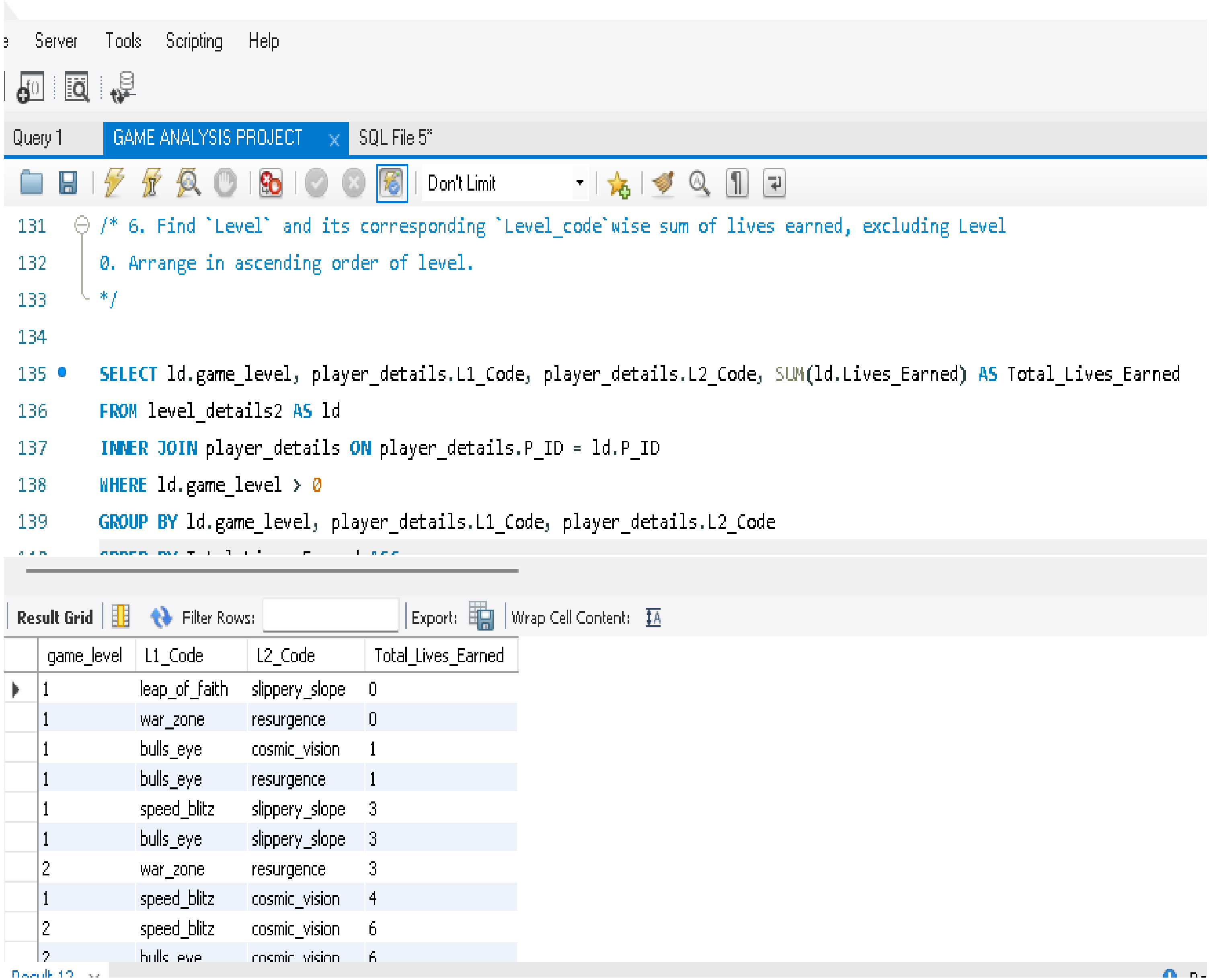
FROM LEVEL_DETAILS2 AS LD

INNER JOIN PLAYER_DETAILS ON PLAYER_DETAILS.P_ID = LD.P_ID

WHERE LD.GAME_LEVEL > 0

GROUP BY LD.GAME_LEVEL, PLAYER_DETAILS.L1_CODE, PLAYER_DETAILS.L2_CODE

ORDER BY TOTAL_LIVES_EARNED ASC;



```
/* 6. Find `Level` and its corresponding `Level_code`wise sum of lives earned, excluding Level
0. Arrange in ascending order of level.
*/

SELECT ld.game_level, player_details.L1_Code, player_details.L2_Code, SUM(ld.Lives_Earned) AS Total_Lives_Earned
FROM level_details2 AS ld
INNER JOIN player_details ON player_details.P_ID = ld.P_ID
WHERE ld.game_level > 0
GROUP BY ld.game_level, player_details.L1_Code, player_details.L2_Code
```

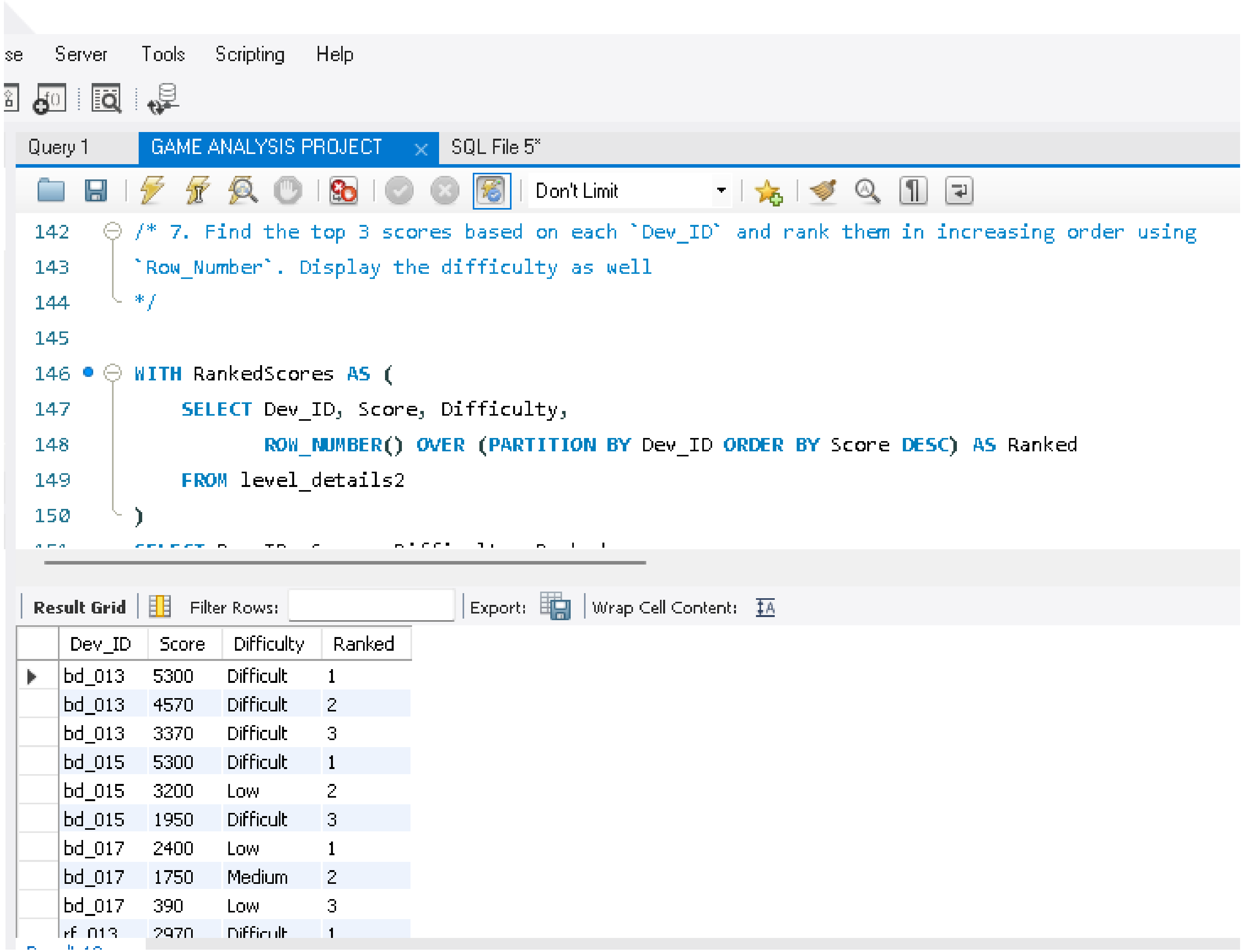
game_level	L1_Code	L2_Code	Total_Lives_Earned
1	leap_of_faith	slippery_slope	0
1	war_zone	resurgence	0
1	bulls_eye	cosmic_vision	1
1	bulls_eye	resurgence	1
1	speed_blitz	slippery_slope	3
1	bulls_eye	slippery_slope	3
2	war_zone	resurgence	3
1	speed_blitz	cosmic_vision	4
2	speed_blitz	cosmic_vision	6
2	bulls_eye	cosmic_vision	6

DATA ANALYSIS

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, SCORE,
DIFFICULTY,

           ROW_NUMBER() OVER
(PARTITION BY DEV_ID ORDER BY
SCORE DESC) AS RANKED

    FROM LEVEL_DETAILS2

)

SELECT DEV_ID, SCORE,
DIFFICULTY, RANKED

FROM RANKEDSCORES

WHERE RANKED <= 3;
```



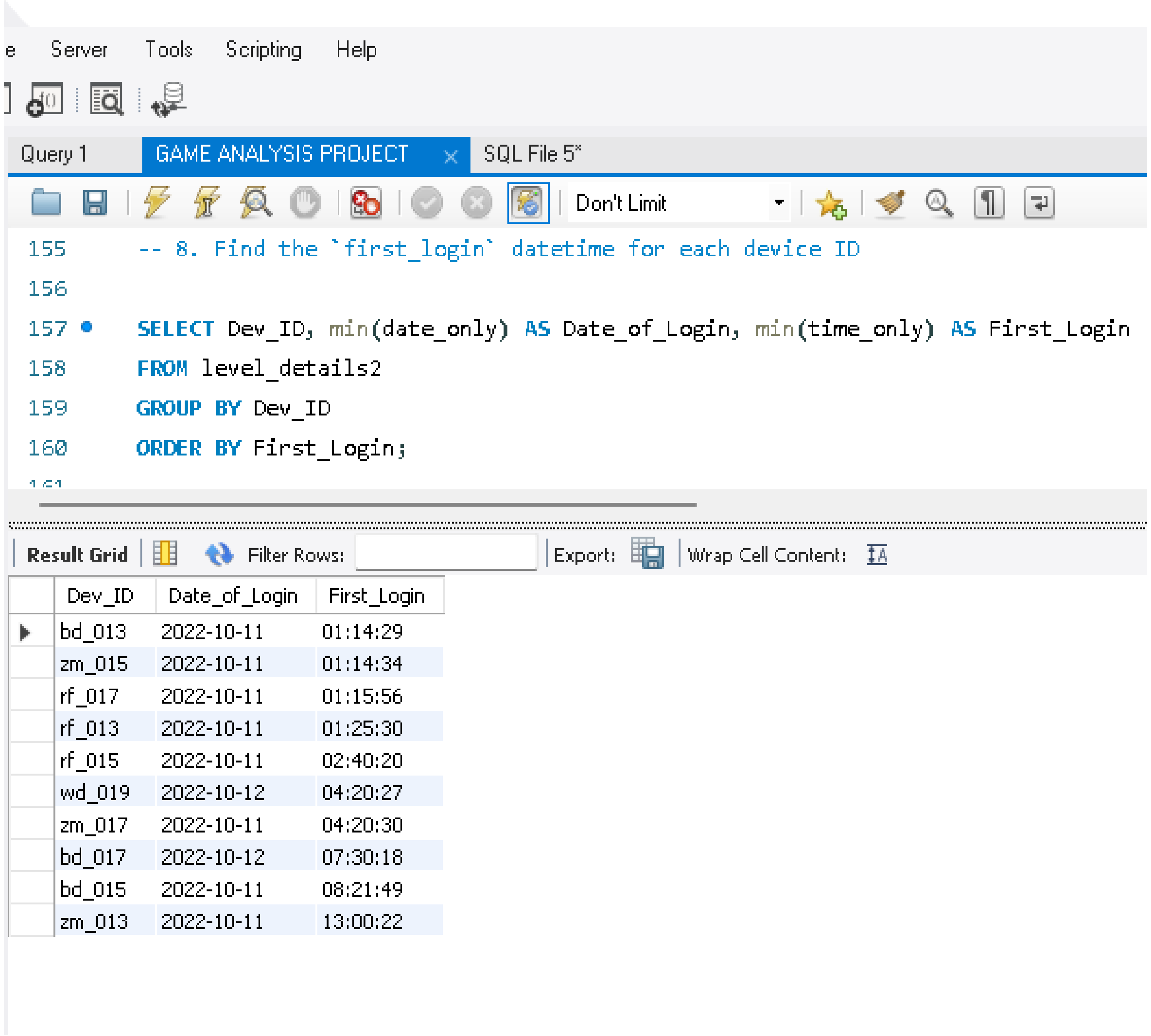
```
/* 7. Find the top 3 scores based on each `Dev_ID` and rank them in increasing order using
`Row_Number`. Display the difficulty as well
*/

WITH RankedScores AS (
    SELECT Dev_ID, Score, Difficulty,
           ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY Score DESC) AS Ranked
    FROM level_details2
)
```

Dev_ID	Score	Difficulty	Ranked
bd_013	5300	Difficult	1
bd_013	4570	Difficult	2
bd_013	3370	Difficult	3
bd_015	5300	Difficult	1
bd_015	3200	Low	2
bd_015	1950	Difficult	3
bd_017	2400	Low	1
bd_017	1750	Medium	2
bd_017	390	Low	3
rf_013	2970	Difficult	1

DATA ANALYSIS

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

```
SELECT DEV_ID, MIN(DATE_ONLY) AS DATE_OF_LOGIN, MIN(TIME_ONLY) AS FIRST_LOGIN

FROM LEVEL_DETAILS2

GROUP BY DEV_ID

ORDER BY FIRST_LOGIN;
```



Don't Limit

```
-- 8. Find the `first_login` datetime for each device ID

SELECT Dev_ID, min(date_only) AS Date_of_Login, min(time_only) AS First_Login
FROM level_details2
GROUP BY Dev_ID
ORDER BY First_Login;
```

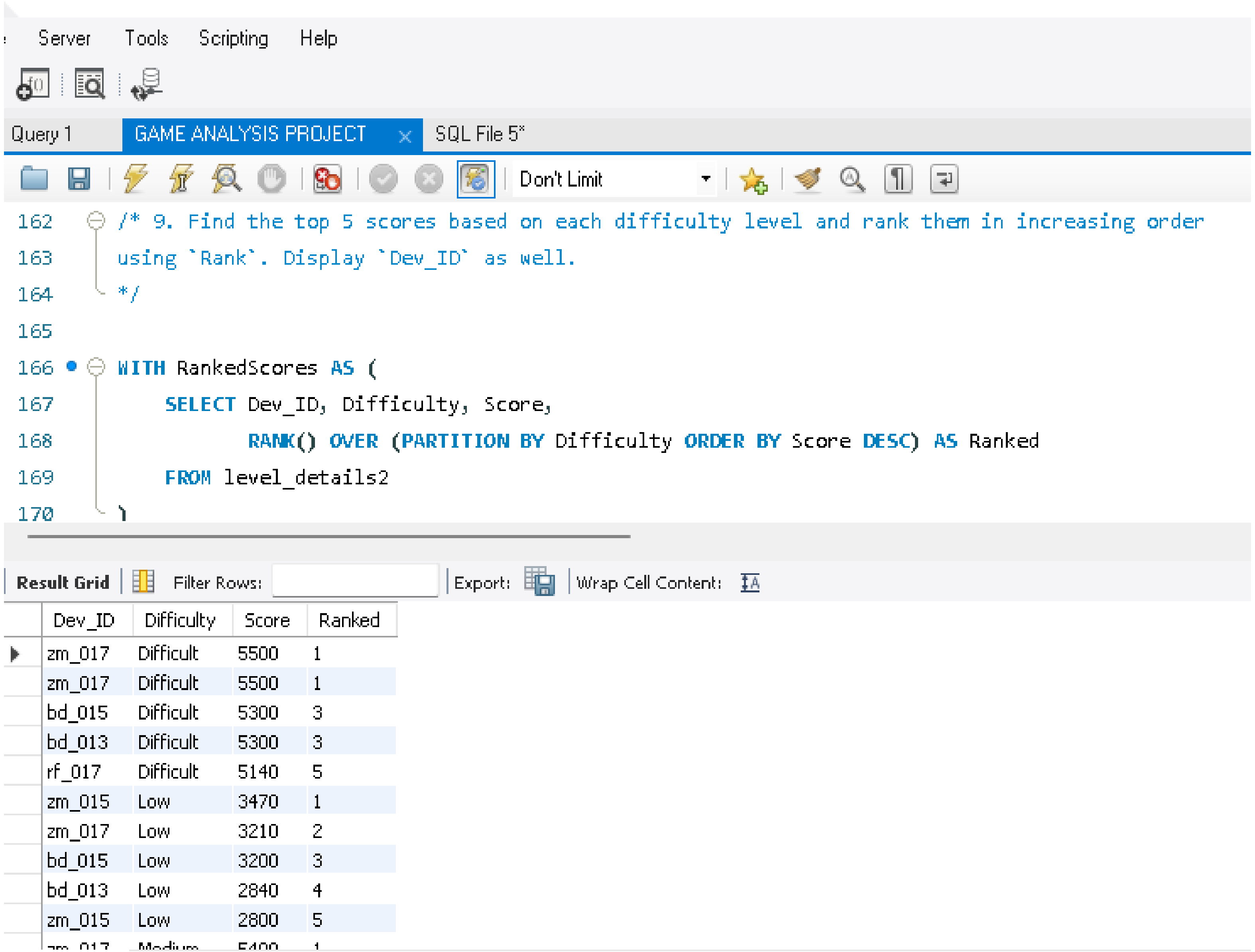
Dev_ID	Date_of_Login	First_Login
bd_013	2022-10-11	01:14:29
zm_015	2022-10-11	01:14:34
rf_017	2022-10-11	01:15:56
rf_013	2022-10-11	01:25:30
rf_015	2022-10-11	02:40:20
wd_019	2022-10-12	04:20:27
zm_017	2022-10-11	04:20:30
bd_017	2022-10-12	07:30:18
bd_015	2022-10-11	08:21:49
zm_013	2022-10-11	13:00:22

DATA ANALYSIS

9. FIND THE TOP 5 SCORES BASED ON EACH DIFFICULTY LEVEL AND RANK THEM IN INCREASING ORDER

USING `RANK`. DISPLAY `DEV_ID` AS WELL.

```
WITH RANKEDSCORES AS (

    SELECT DEV_ID, DIFFICULTY,
SCORE,

           RANK() OVER
(PARTITION BY DIFFICULTY ORDER
BY SCORE DESC) AS RANKED

    FROM LEVEL_DETAILS2

)

SELECT DEV_ID, DIFFICULTY,
SCORE, RANKED

FROM RANKEDSCORES

WHERE RANKED <= 5;
```



```
/* 9. Find the top 5 scores based on each difficulty level and rank them in increasing order
using `Rank`. Display `Dev_ID` as well.
*/

WITH RankedScores AS (
    SELECT Dev_ID, Difficulty, Score,
           RANK() OVER (PARTITION BY Difficulty ORDER BY Score DESC) AS Ranked
    FROM level_details2
)
```

Dev_ID	Difficulty	Score	Ranked
zm_017	Difficult	5500	1
zm_017	Difficult	5500	1
bd_015	Difficult	5300	3
bd_013	Difficult	5300	3
rf_017	Difficult	5140	5
zm_015	Low	3470	1
zm_017	Low	3210	2
bd_015	Low	3200	3
bd_013	Low	2840	4
zm_015	Low	2800	5
zm_017	Medium	5400	1

DATA ANALYSIS

10. FIND THE DEVICE ID THAT IS FIRST LOGGED IN (BASED ON `START_DATETIME`) FOR EACH PLAYER

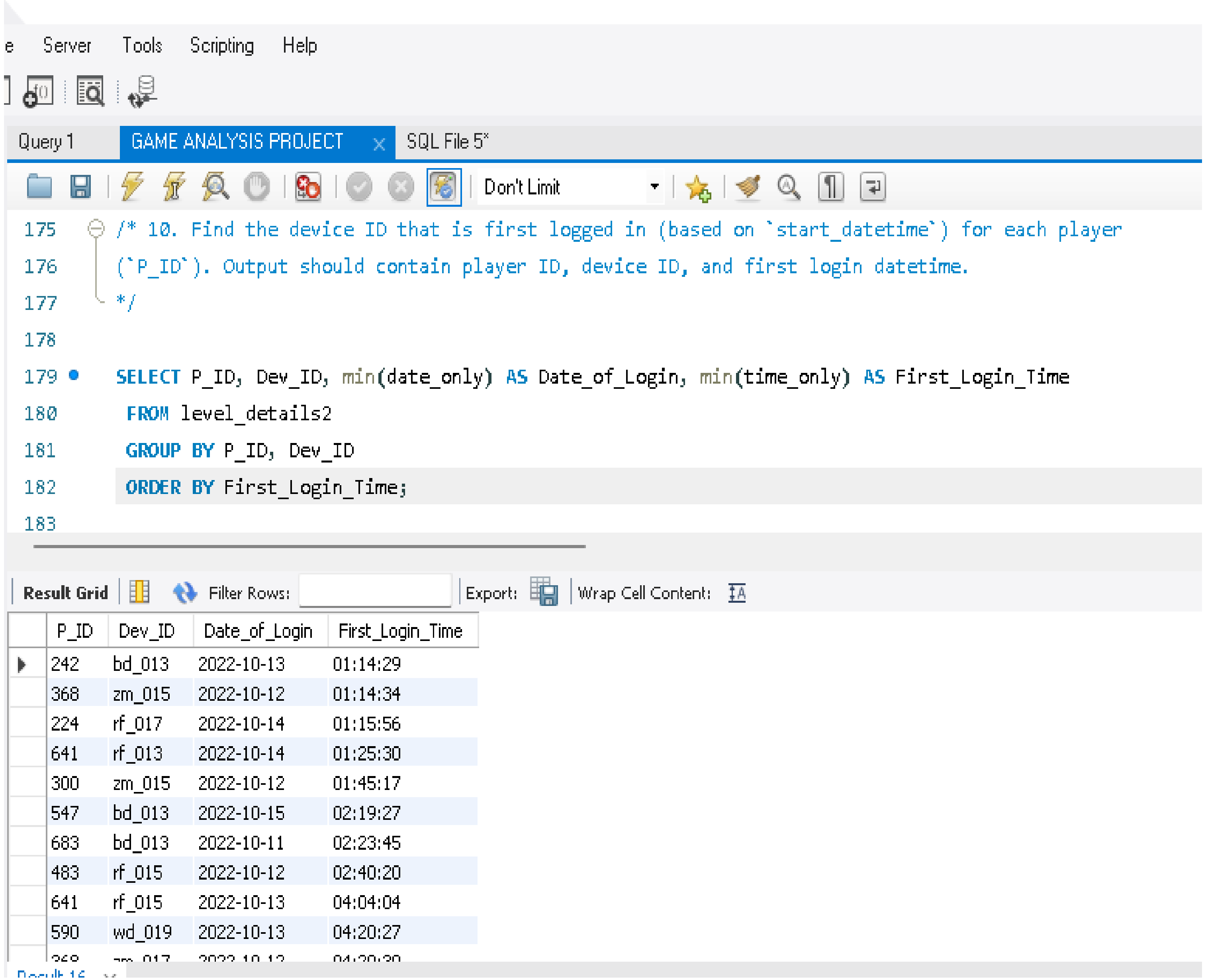
(`P_ID`). OUTPUT SHOULD CONTAIN PLAYER ID, DEVICE ID, AND FIRST LOGIN DATETIME.

SELECT P_ID, DEV_ID, MIN(DATE_ONLY) AS DATE_OF_LOGIN, MIN(TIME_ONLY) AS FIRST_LOGIN_TIME

FROM LEVEL_DETAILS2

GROUP BY P_ID, DEV_ID

ORDER BY FIRST_LOGIN_TIME;



```
/* 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.
*/

SELECT P_ID, Dev_ID, min(date_only) AS Date_of_Login, min(time_only) AS First_Login_Time
 FROM level_details2
 GROUP BY P_ID, Dev_ID
 ORDER BY First_Login_Time;
```

P_ID	Dev_ID	Date_of_Login	First_Login_Time
242	bd_013	2022-10-13	01:14:29
368	zm_015	2022-10-12	01:14:34
224	rf_017	2022-10-14	01:15:56
641	rf_013	2022-10-14	01:25:30
300	zm_015	2022-10-12	01:45:17
547	bd_013	2022-10-15	02:19:27
683	bd_013	2022-10-11	02:23:45
483	rf_015	2022-10-12	02:40:20
641	rf_015	2022-10-13	04:04:04
590	wd_019	2022-10-13	04:20:27
368	zm_017	2022-10-12	04:20:30

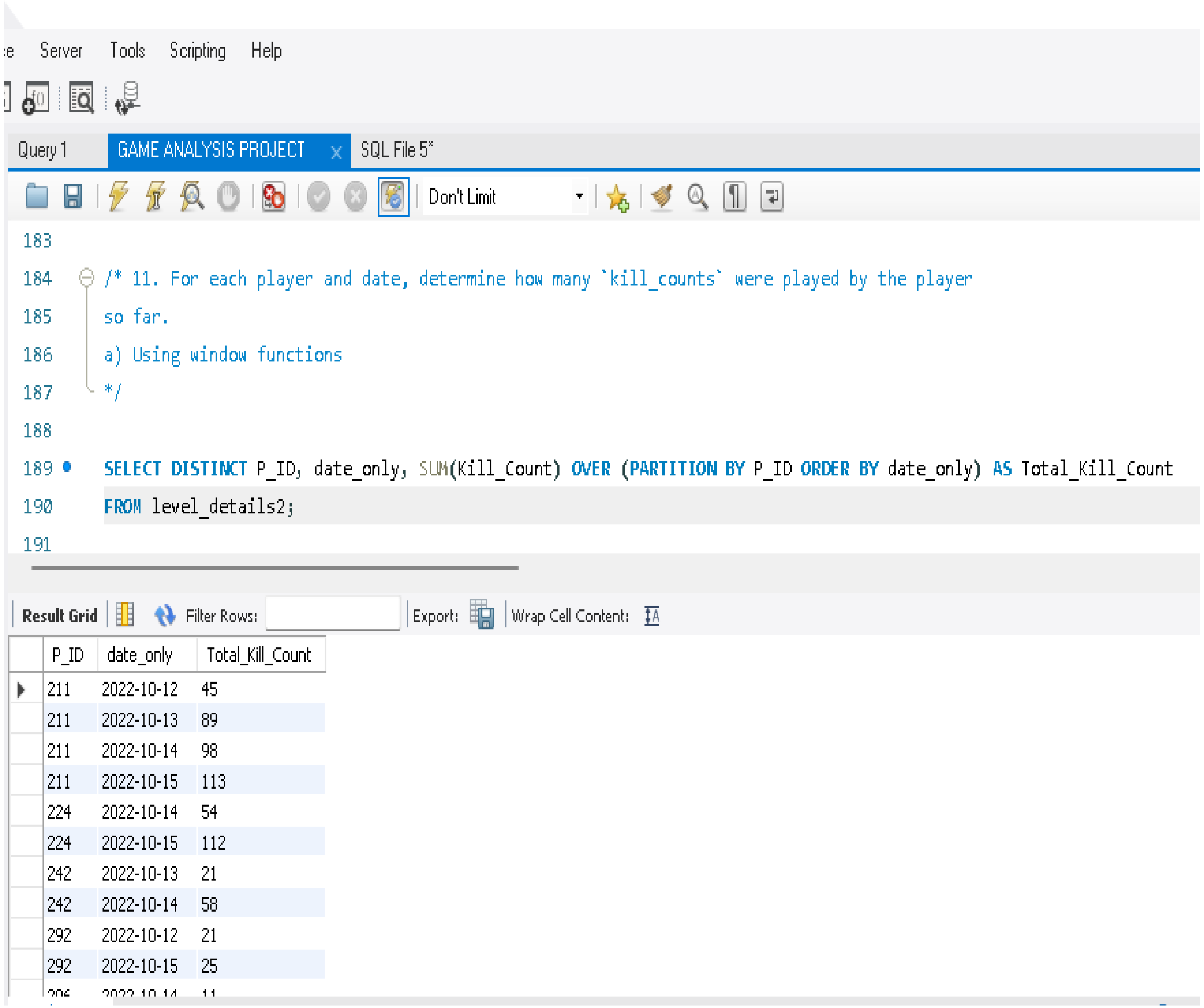
DATA ANALYSIS

11. FOR EACH PLAYER AND DATE, DETERMINE HOW MANY `KILL_COUNTS` WERE PLAYED BY THE PLAYER

SO FAR.

A) USING WINDOW FUNCTIONS

```
SELECT DISTINCT P_ID, DATE_ONLY, SUM(KILL_COUNT) OVER (PARTITION BY P_ID ORDER BY DATE_ONLY) AS TOTAL_KILL_COUNT

FROM LEVEL_DETAILS2;
```



```
/* 11. For each player and date, determine how many `kill_counts` were played by the player
so far.
a) Using window functions
*/

SELECT DISTINCT P_ID, date_only, SUM(Kill_Count) OVER (PARTITION BY P_ID ORDER BY date_only) AS Total_Kill_Count
FROM level_details2;
```

P_ID	date_only	Total_Kill_Count
211	2022-10-12	45
211	2022-10-13	89
211	2022-10-14	98
211	2022-10-15	113
224	2022-10-14	54
224	2022-10-15	112
242	2022-10-13	21
242	2022-10-14	58
292	2022-10-12	21
292	2022-10-15	25
296	2022-10-14	11

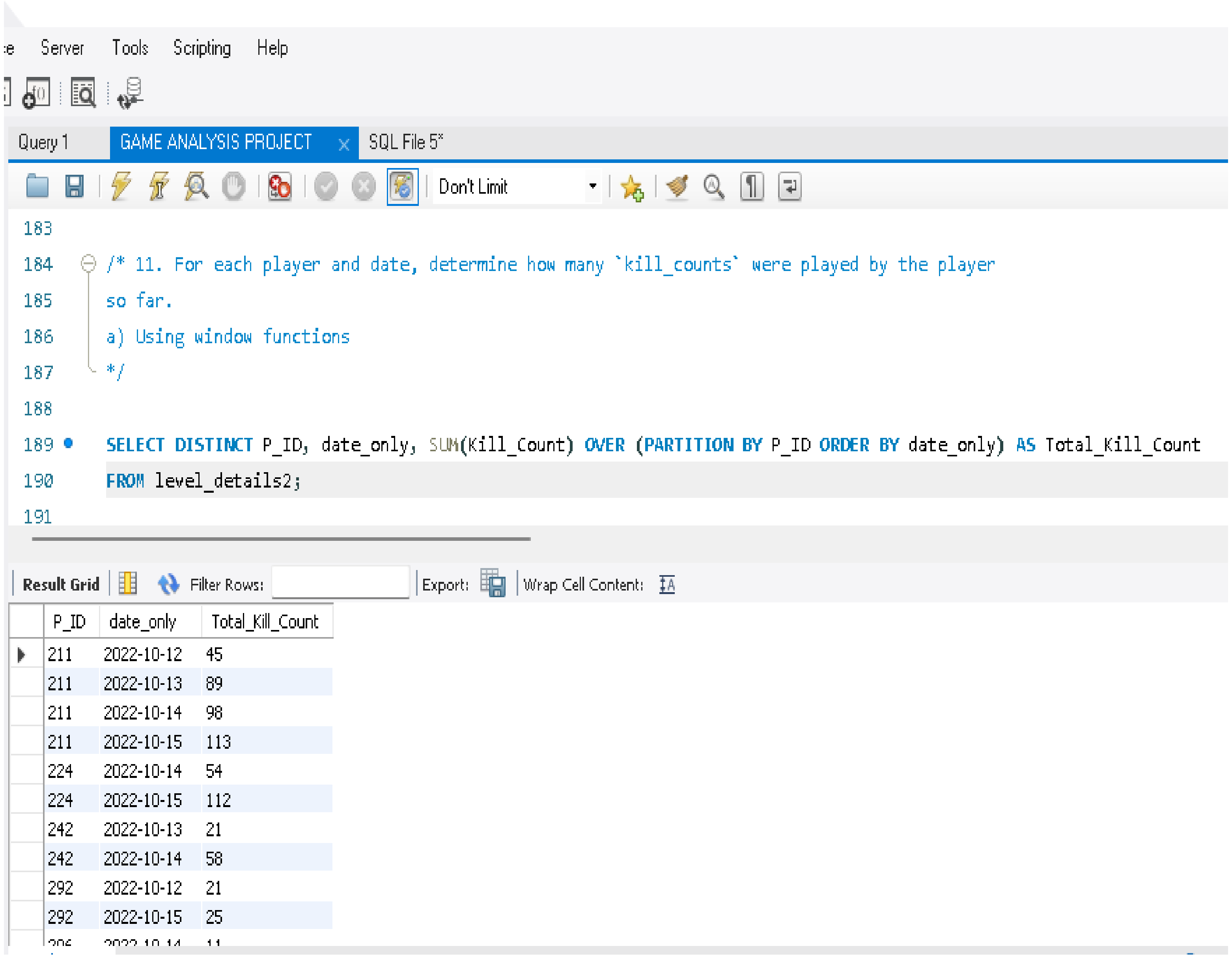
DATA ANALYSIS

11. FOR EACH PLAYER AND DATE, DETERMINE HOW MANY `KILL_COUNTS` WERE PLAYED BY THE PLAYER

SO FAR.

B) WITHOUT WINDOW FUNCTIONS

```
SELECT DISTINCT

    T1.P_ID, T1.DATE_ONLY,

    (SELECT
SUM(T2.KILL_COUNT)

     FROM LEVEL_DETAILS2 T2

     WHERE T1.P_ID = T2.P_ID
AND T1.DATE_ONLY >=
T2.DATE_ONLY) AS
TOTAL_KILL_COUNT

FROM

    LEVEL_DETAILS2 T1

ORDER BY

    T1.P_ID, T1.DATE_ONLY;
```



```
/* 11. For each player and date, determine how many `kill_counts` were played by the player
so far.
a) Using window functions
*/

SELECT DISTINCT P_ID, date_only, SUM(Kill_Count) OVER (PARTITION BY P_ID ORDER BY date_only) AS Total_Kill_Count
FROM level_details2;
```

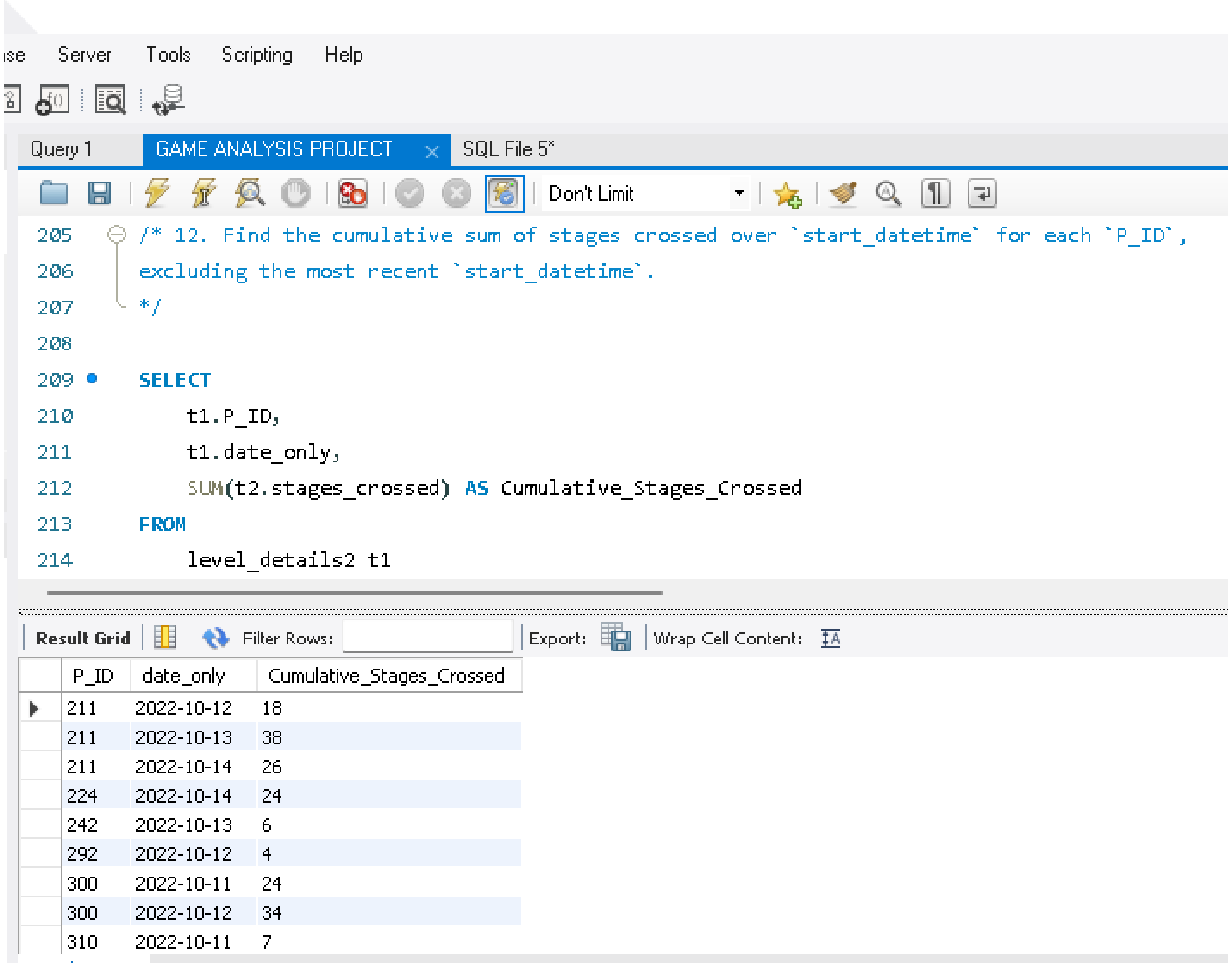
P_ID	date_only	Total_Kill_Count
211	2022-10-12	45
211	2022-10-13	89
211	2022-10-14	98
211	2022-10-15	113
224	2022-10-14	54
224	2022-10-15	112
242	2022-10-13	21
242	2022-10-14	58
292	2022-10-12	21
292	2022-10-15	25
296	2022-10-14	11

DATA ANALYSIS

12. FIND THE CUMULATIVE SUM OF STAGES CROSSED OVER `START_DATETIME` FOR EACH `P_ID`,

EXCLUDING THE MOST RECENT `START_DATETIME`.

```
SELECT

    T1.P_ID, T1.DATE_ONLY,

    SUM(T2.STAGES_CROSSED) AS
CUMULATIVE_STAGES_CROSSED

FROM LEVEL_DETAILS2 T1

JOIN LEVEL_DETAILS2 T2 ON
T1.P_ID = T2.P_ID AND
T1.DATE_ONLY >= T2.DATE_ONLY

GROUP BY T1.P_ID, T1.DATE_ONLY

HAVING T1.DATE_ONLY < (SELECT
MAX(DATE_ONLY) FROM
LEVEL_DETAILS2 WHERE P_ID =
T1.P_ID)

ORDER BY T1.P_ID,
T1.DATE_ONLY;
```



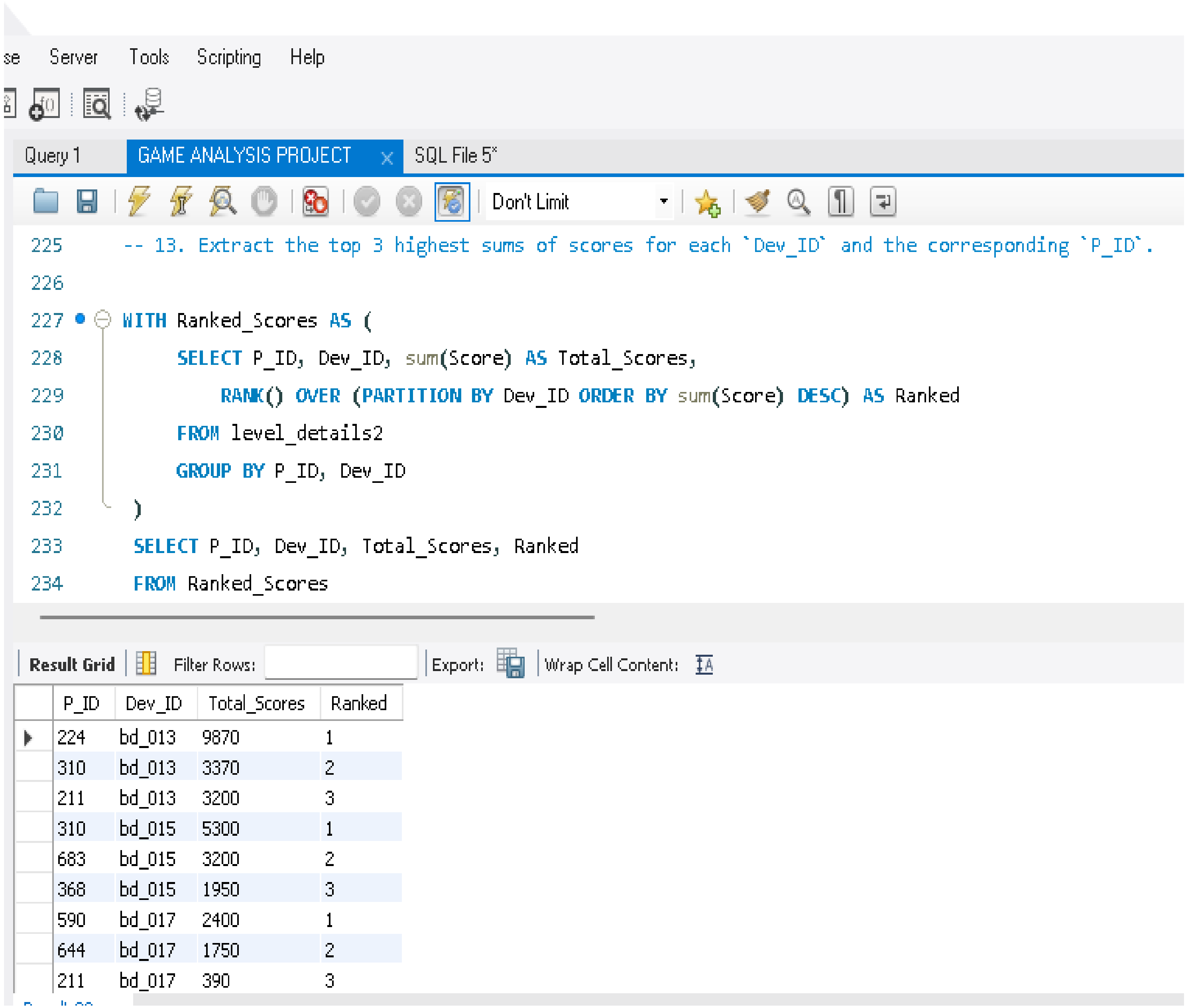
```
/* 12. Find the cumulative sum of stages crossed over `start_datetime` for each `P_ID`,
excluding the most recent `start_datetime`.
*/

SELECT
    t1.P_ID,
    t1.date_only,
    SUM(t2.stages_crossed) AS Cumulative_Stages_Crossed
FROM
    level_details2 t1
```

P_ID	date_only	Cumulative_Stages_Crossed
211	2022-10-12	18
211	2022-10-13	38
211	2022-10-14	26
224	2022-10-14	24
242	2022-10-13	6
292	2022-10-12	4
300	2022-10-11	24
300	2022-10-12	34
310	2022-10-11	7

DATA ANALYSIS

13. EXTRACT THE TOP 3 HIGHEST SUMS OF SCORES FOR EACH `DEV_ID` AND THE CORRESPONDING `P_ID`.

```
WITH RANKED_SCORES AS (

     SELECT P_ID, DEV_ID, 
SUM(SCORE) AS TOTAL_SCORES,

         RANK() OVER 
(PARTITION BY DEV_ID ORDER BY 
SUM(SCORE) DESC) AS RANKED

     FROM LEVEL_DETAILS2

     GROUP BY P_ID, DEV_ID

)

SELECT P_ID, DEV_ID, 
TOTAL_SCORES, RANKED

FROM RANKED_SCORES

WHERE RANKED <= 3;
```



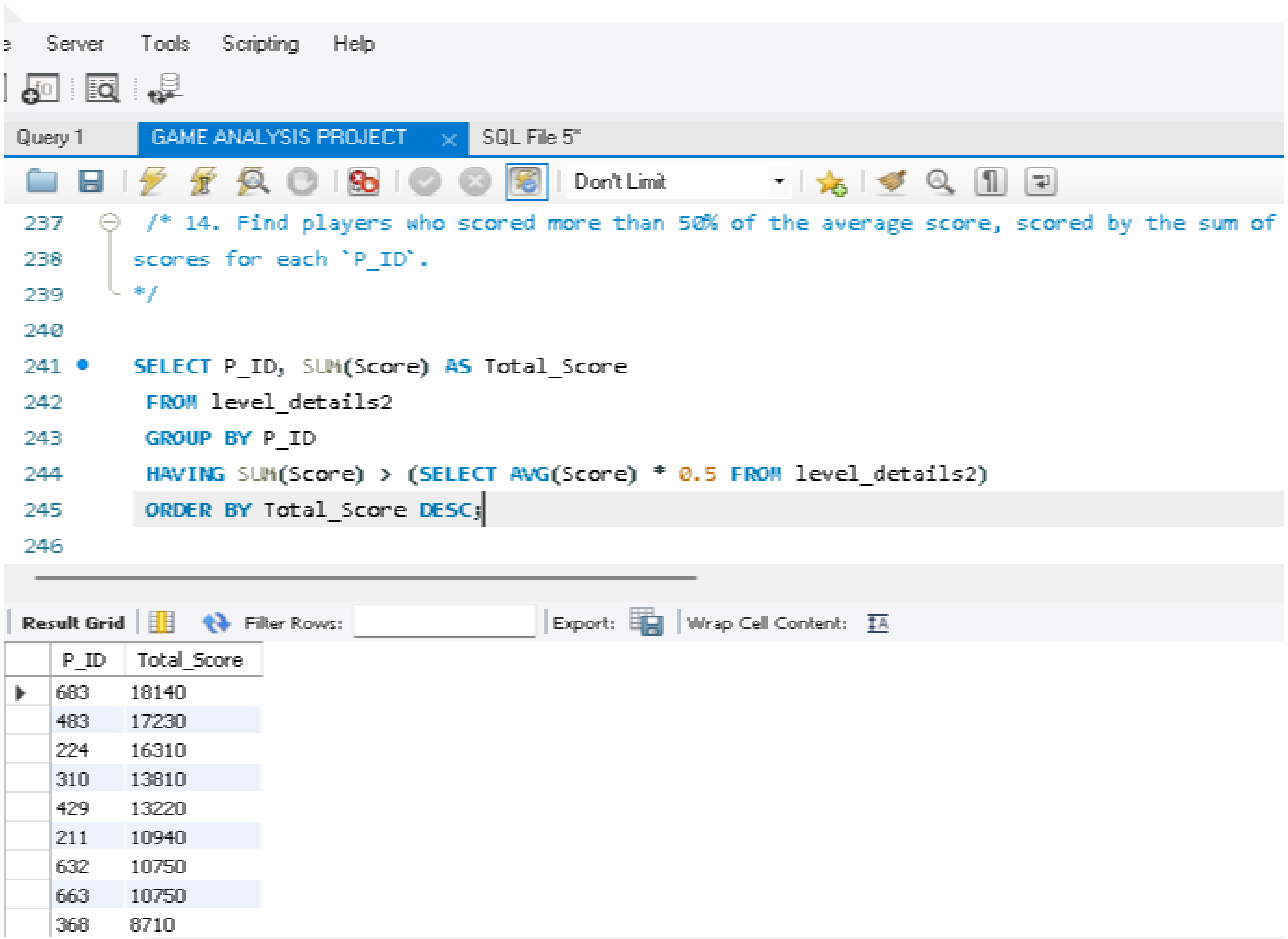
```
-- 13. Extract the top 3 highest sums of scores for each `Dev_ID` and the corresponding `P_ID`.

WITH Ranked_Scores AS (
    SELECT P_ID, Dev_ID, sum(Score) AS Total_Scores,
        RANK() OVER (PARTITION BY Dev_ID ORDER BY sum(Score) DESC) AS Ranked
    FROM level_details2
    GROUP BY P_ID, Dev_ID
)
SELECT P_ID, Dev_ID, Total_Scores, Ranked
FROM Ranked_Scores
```

P_ID	Dev_ID	Total_Scores	Ranked
224	bd_013	9870	1
310	bd_013	3370	2
211	bd_013	3200	3
310	bd_015	5300	1
683	bd_015	3200	2
368	bd_015	1950	3
590	bd_017	2400	1
644	bd_017	1750	2
211	bd_017	390	3

DATA ANALYSIS

14. FIND PLAYERS WHO SCORED MORE THAN 50% OF THE AVERAGE SCORE, SCORED BY THE SUM OF SCORES FOR EACH `P_ID`.

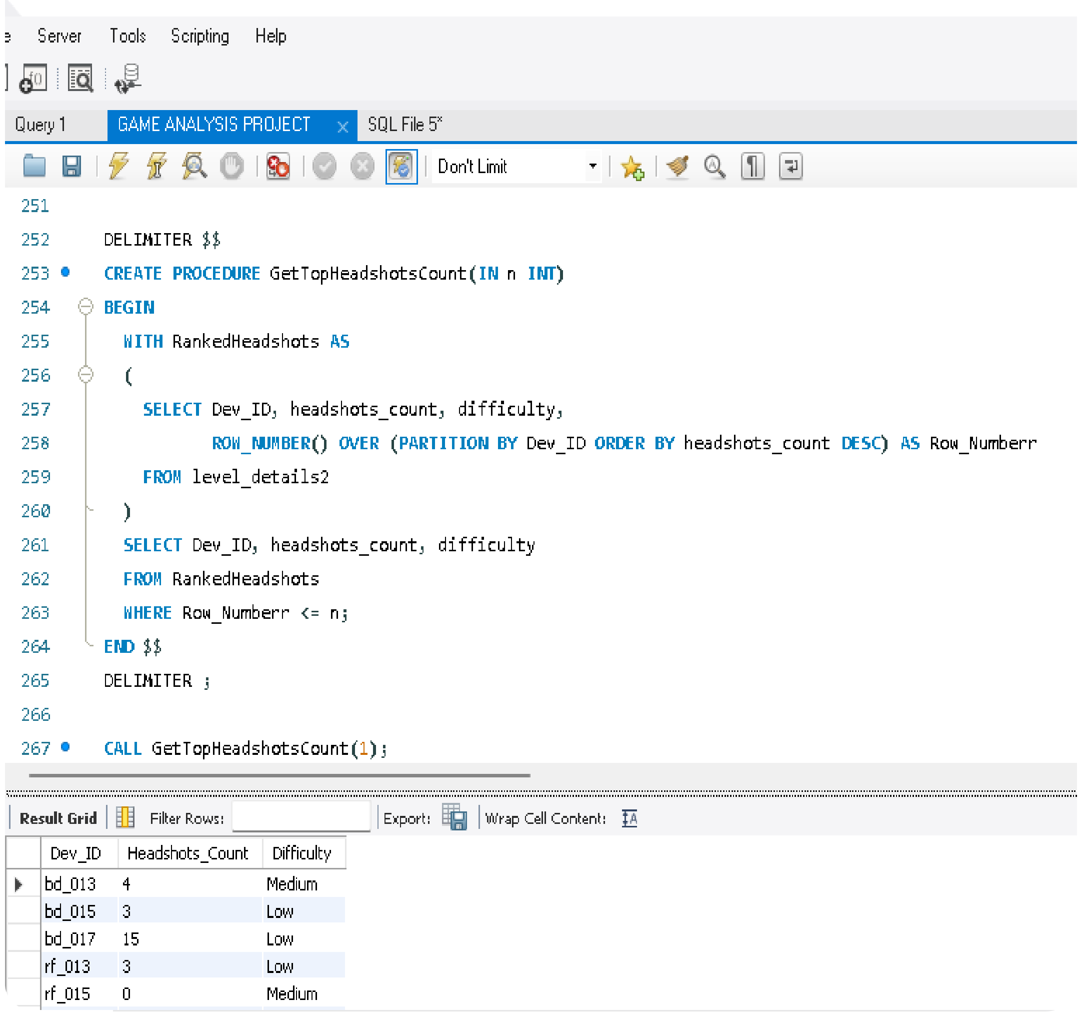
```
SELECT P_ID, SUM(SCORE) AS TOTAL_SCORE

FROM LEVEL_DETAILS2

GROUP BY P_ID

HAVING SUM(SCORE) > (SELECT AVG(SCORE) * 0.5 FROM LEVEL_DETAILS2)

ORDER BY TOTAL_SCORE DESC;
```



```
/* 14. Find players who scored more than 50% of the average score, scored by the sum of
scores for each `P_ID`.
*/

SELECT P_ID, SUM(Score) AS Total_Score
 FROM level_details2
 GROUP BY P_ID
 HAVING SUM(Score) > (SELECT AVG(Score) * 0.5 FROM level_details2)
 ORDER BY Total_Score DESC;
```

P_ID	Total_Score
683	18140
483	17230
224	16310
310	13810
429	13220
211	10940
632	10750
663	10750
368	8710

DATA ANALYSIS



```
DELIMITER $$
CREATE PROCEDURE GetTopHeadshotsCount(IN n INT)
BEGIN
  WITH RankedHeadshots AS
  (
    SELECT Dev_ID, headshots_count, difficulty,
           ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY headshots_count DESC) AS Row_Numberr
    FROM level_details2
  )
  SELECT Dev_ID, headshots_count, difficulty
  FROM RankedHeadshots
  WHERE Row_Numberr <= n;
END $$
DELIMITER ;

CALL GetTopHeadshotsCount(1);
```

Dev_ID	Headshots_Count	Difficulty
bd_013	4	Medium
bd_015	3	Low
bd_017	15	Low
rf_013	3	Low
rf_015	0	Medium

- 15. CREATE A STORED PROCEDURE TO FIND THE TOP `N` `HEADSHOTS_COUNT` BASED ON EACH `DEV_ID`
- AND RANK THEM IN INCREASING ORDER USING `ROW_NUMBER`. DISPLAY THE DIFFICULTY AS WELL.

- DELIMITER $$
- CREATE PROCEDURE GETTOPHEADSHOTSCOUNT(IN N INT)
- BEGIN
- WITH RANKEDHEADSHOTS AS
- (
- SELECT DEV_ID, HEADSHOTS_COUNT, DIFFICULTY,
- ROW_NUMBER() OVER (PARTITION BY DEV_ID ORDER BY HEADSHOTS_COUNT DESC) AS ROW_NUMBERR
- FROM LEVEL_DETAILS2
-)
- SELECT DEV_ID, HEADSHOTS_COUNT, DIFFICULTY
- FROM RANKEDHEADSHOTS
- WHERE ROW_NUMBERR <= N;
- END $$
- DELIMITER ;

- CALL GETTOPHEADSHOTSCOUNT(1);

INSIGHTS

- Focus on improving player engagement for low and medium difficulty levels, as the total stages crossed data shows a significant drop compared to the difficult level for players using zm_series devices.
- Identify and analyze the players who have played games on multiple days. These players are likely more engaged and can provide valuable insights into factors that encourage consistent gameplay.
- Optimize the game mechanics or levels where the kill counts exceed the average for medium difficulty. This could help retain players who find the medium difficulty too easy.
- Analyze the level codes and corresponding difficulty levels where the most lives are earned. These levels or mechanics could be further enhanced or replicated in other levels to increase player engagement.
- Identify the top-performing device IDs based on scores and difficulty levels. Study the characteristics of these devices and their users to understand what contributes to their success.

INSIGHTS

- Analyze the first login data to identify potential patterns or trends in player onboarding and engagement. This could help optimize the onboarding process and improve player retention from the start.
- Study the players who consistently score above the average. These players could provide insights into factors that contribute to higher engagement and better performance.
- Examine the cumulative stages crossed data to identify potential drop-off points where players tend to lose interest or face challenges. These could be opportunities for game improvements or targeted player support.
- Analyze the top-scoring players for each device ID to understand the preferences and behaviors of high-performing players. This could inform strategies for encouraging similar gameplay patterns among other players.
- Consider implementing features or incentives to encourage consistent gameplay, as the data on kill counts per date could reveal patterns of player engagement or dropoff over time.

RECOMMENDATIONS

1. Difficulty Level Optimization:

- Conduct a thorough review of the low and medium difficulty levels to identify areas for improvement. Analyze player feedback, gameplay data, and performance metrics to pinpoint potential pain points or areas where players lose interest.

- Consider adjusting the difficulty curve, introducing new mechanics, or enhancing existing gameplay elements to make these levels more engaging and rewarding.

2. Player Segmentation and Targeted Engagement:

- Segment players based on their engagement levels, performance, and preferences. Identify highly engaged players, casual players, and those at risk of churn.

- Develop targeted strategies for each segment, such as personalized challenges, rewards, or in-game events, to enhance their gameplay experience and encourage continued engagement.

RECOMMENDATIONS

3. Onboarding and Retention Strategies:

- Analyze the first login data and player behavior during the initial gameplay stages to optimize the onboarding process.

- Implement tutorials, guidance, or incentives that help new players understand the game mechanics and progress smoothly through the early levels.

- Identify potential dropoff points and introduce interventions or adjustments to maintain player interest and prevent churn.

4. High-Performance Player Analysis:

- Conduct in-depth interviews or surveys with top-performing players to understand their motivations, preferences, and gameplay strategies.

- Analyze the characteristics of devices and device IDs associated with high-performing players to identify potential hardware or software factors contributing to their success.

- Leverage these insights to develop features, updates, or marketing campaigns that cater to the preferences of high-performing players.

RECOMMENDATIONS

5. Community Engagement and Social Features:

- Evaluate the potential for introducing social features or community engagement opportunities within the game.

- Foster a sense of community and competition among players, which can increase engagement and retention.

- Encourage players to share their experiences, strategies, and achievements, creating a more immersive and interactive gaming environment.

6. Continuous Data Monitoring and Iteration:

- Implement robust data tracking and analysis processes to monitor player behavior, performance metrics, and engagement levels on an ongoing basis.

- Continuously iterate and refine the game based on these insights, introducing updates, adjustments, or new features to maintain player interest and satisfaction.

RECOMMENDATIONS

7. Cross-Platform and Accessibility Considerations:

- Evaluate the game's performance and accessibility across various platforms and devices, ensuring a consistent and optimized experience for all players.

- Address any platform-specific issues or limitations that may impact gameplay or engagement.

8. Collaboration and Player Feedback:

- Foster open communication channels with players to gather feedback, suggestions, and insights.

- Collaborate with the game development team to prioritize and implement player-requested features or improvements, fostering a sense of community involvement and co-creation.

Summary

The gaming project aimed to decode player behavior through data analysis. Key insights include:

1. Optimizing low and medium difficulty levels to improve engagement, as players using zm_series devices showed lower stage completion rates at these levels.

2. Identifying and analyzing highly engaged players who played on multiple days, to understand factors encouraging consistent gameplay.

4. Analyzing first login data and cumulative stages crossed to optimize onboarding and identify potential drop-off points.

5. Considering social features and community engagement to foster player interaction and increase retention.

6. Implementing continuous data monitoring and iterative improvements based on player feedback and behavior.

THANKYOU

VICTOR SIDI 👤

LEONARDIZ054@GMAIL.COM ✉