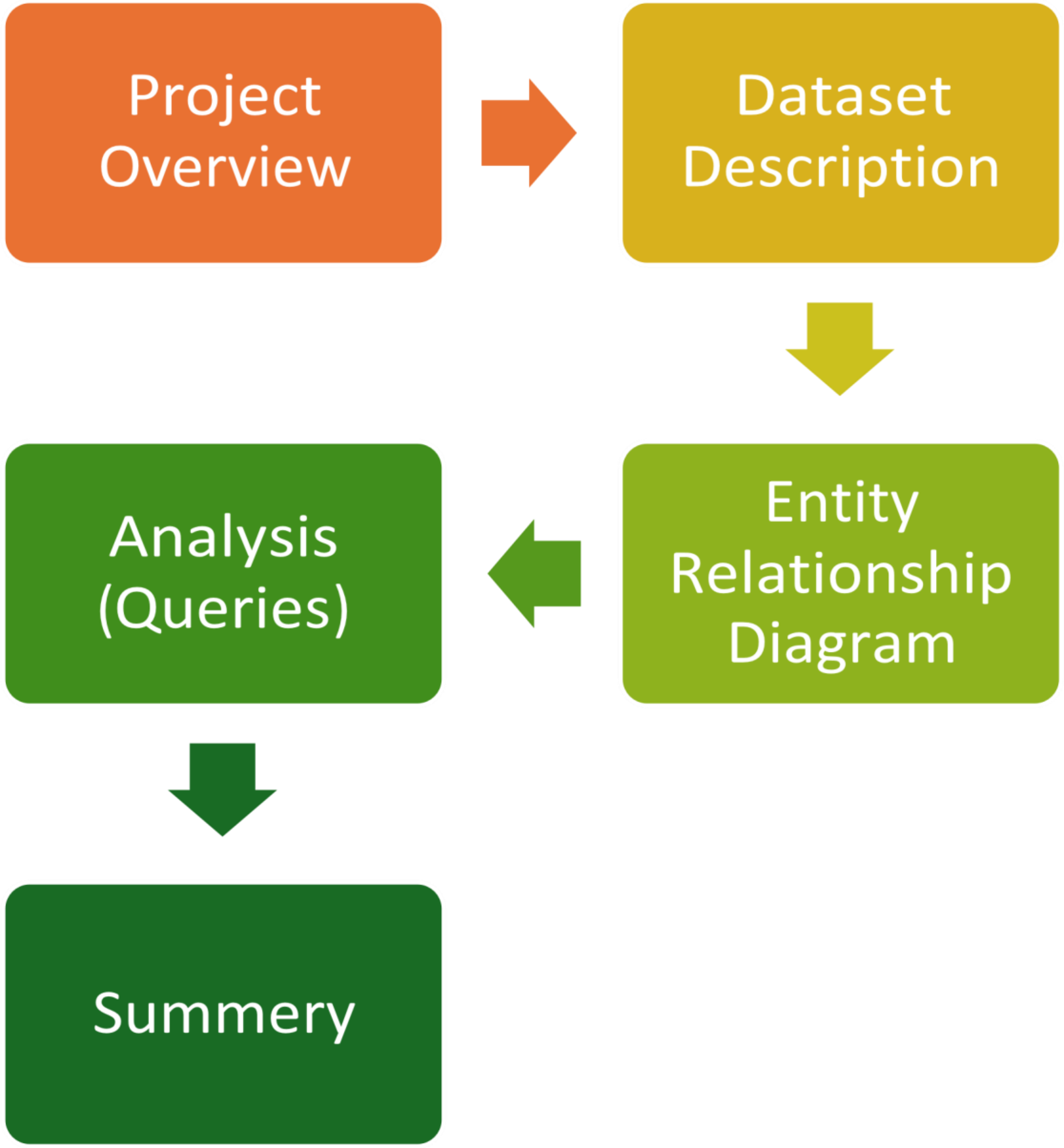
GAME ANALYSIS with SQL

Content



Project Overview

Dataset Description

Entity Relationship Diagram

Analysis (Queries)

Summery

Project Overview

Decode Gaming Behavior” involves analyzing a gaming application’s dataset with ”Player Details” and “Level Details” tables. Its objective is to extract insights into player behavior and performance. Utilizing SQL queries. We aim to understand player engagement, skill progession, and areas for game experience enhancement.

Key questions include player trend, level completion rates, and performance metrics analysis. Our goal is to provide actionable insights for informed decision making in game development.

The project encompasses data exploration, query formulation, result interpretation and data visualization techniques. Through concise presentation.

We facilitate stakeholders’ understanding and decision making in game development and management.

The dataset includes two table : **’Player Details’** and **‘Level Details’**

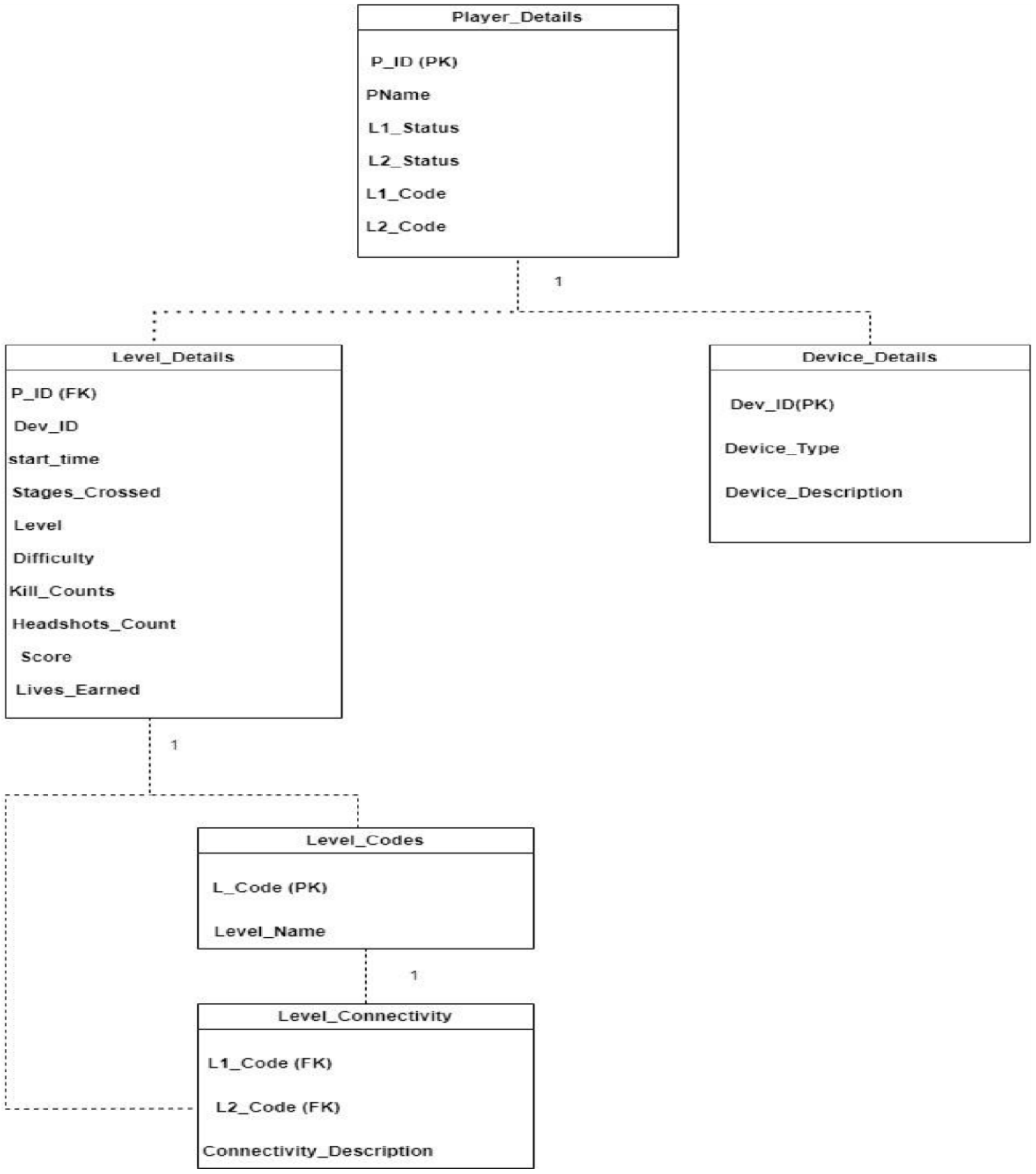
PLAYER DETAILS TABLE:

○ ‘**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

Entity Relationship Diagram



Player_Details

P_ID (PK)

PName

L1_Status

L2_Status

L1_Code

L2_Code

Level_Details

P_ID (FK)

Dev_ID

start_time

Stages_Crossed

Level

Difficulty

Kill_Counts

Headshots_Count

Score

Lives_Earned

Device_Details

Dev_ID(PK)

Device_Type

Device_Description

Level_Codes

L_Code (PK)

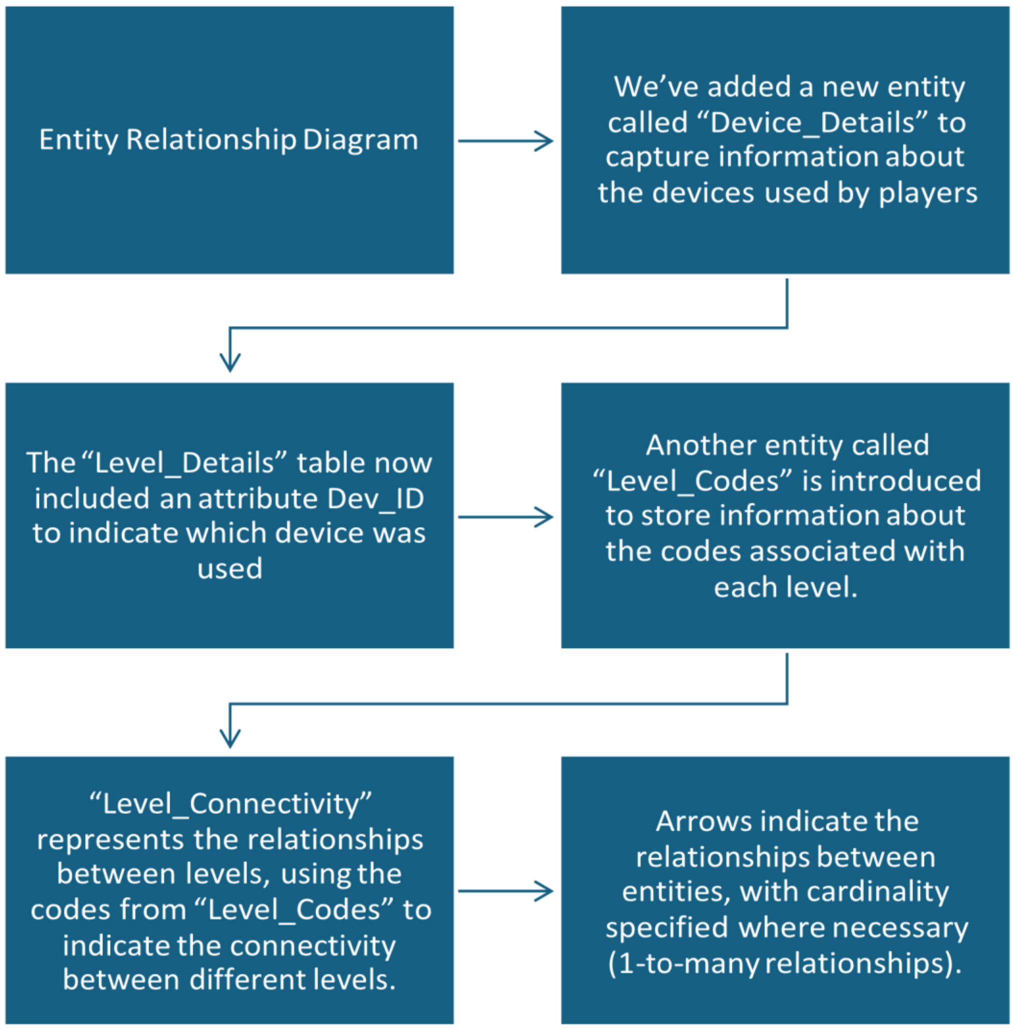
Level_Name

Level_Connectivity

L1_Code (FK)

L2_Code (FK)

Connectivity_Description



Entity Relationship Diagram

We've added a new entity called "Device_Details" to capture information about the devices used by players

The "Level_Details" table now included an attribute Dev_ID to indicate which device was used

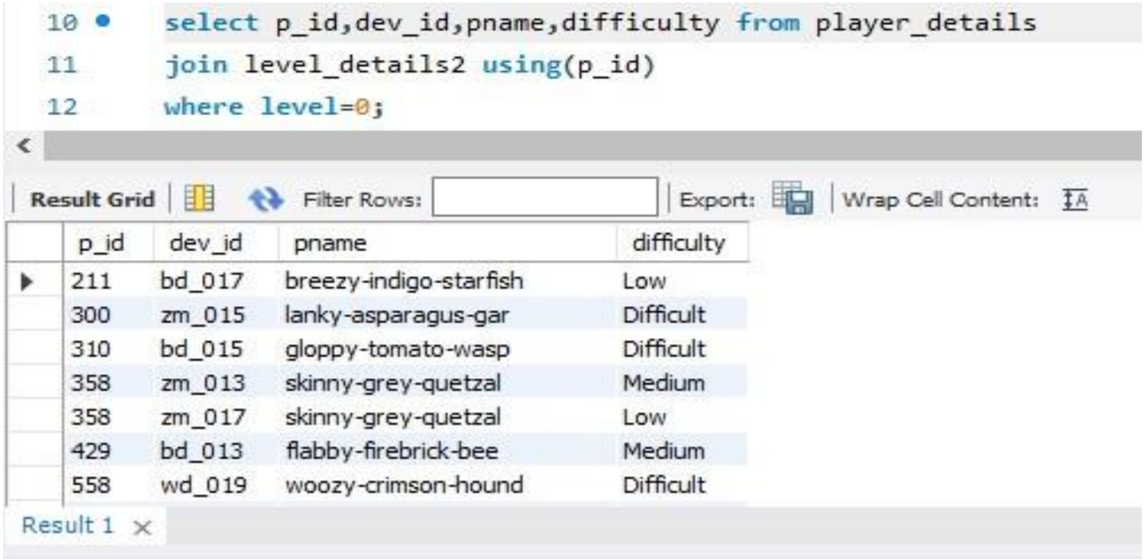
Another entity called "Level_Codes" is introduced to store information about the codes associated with each level.

"Level_Connectivity" represents the relationships between levels, using the codes from "Level_Codes" to indicate the connectivity between different levels.

Arrows indicate the relationships between entities, with cardinality specified where necessary (1-to-many relationships).

Analysis (Queries)

- Query _ 1
- Extract P_ID,Dev_ID,Pname and Difficulty_level of all players at level 0.
- Analysis – it performs an inner join onn the Player ID column between the two tables to retrieve matching records based on the Player ID.



```sql
select p_id,dev_id,pname,difficulty from player_details
join level_details2 using(p_id)
where level=0;
```

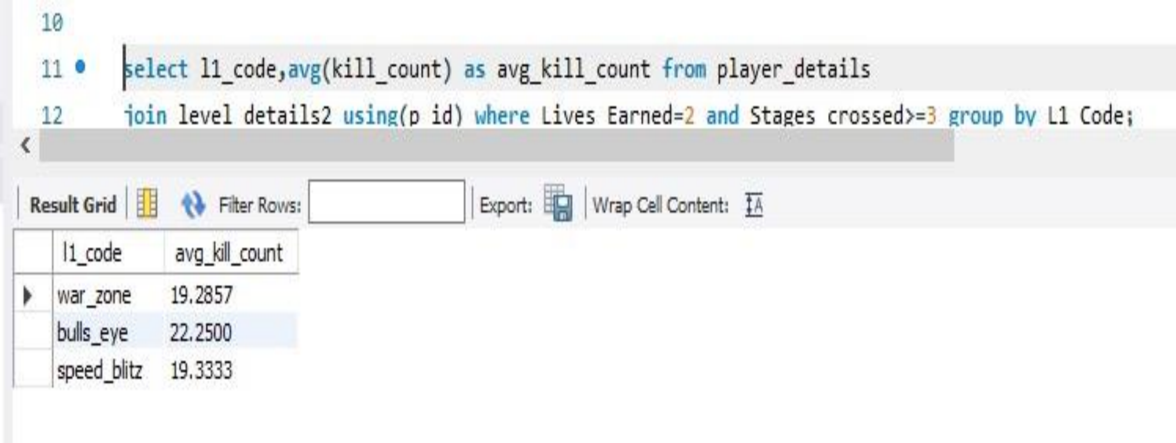
p_id	dev_id	pname	difficulty
211	bd_017	breezy-indigo-starfish	Low
300	zm_015	lanky-asparagus-gar	Difficult
310	bd_015	gloppy-tomato-wasp	Difficult
358	zm_013	skinny-grey-quetzal	Medium
358	zm_017	skinny-grey-quetzal	Low
429	bd_013	flabby-firebrick-bee	Medium
558	wd_019	woozy-crimson-hound	Difficult

Query2:

Final level1_code wise Avg_Kill_Count where lives_earned is 2 and at least 3 stages are crossed

Analysis – it performs an inner join onn the Player ID column between Player_Details and Level_Details tables to retrieve matching records based on the Player ID.

The result is grouped by L1_code.



```
select l1_code,avg(kill_count) as avg_kill_count from player_details
join level_details2 using(p_id) where Lives_Earned=2 and Stages_crossed>=3 group by L1_Code;
```

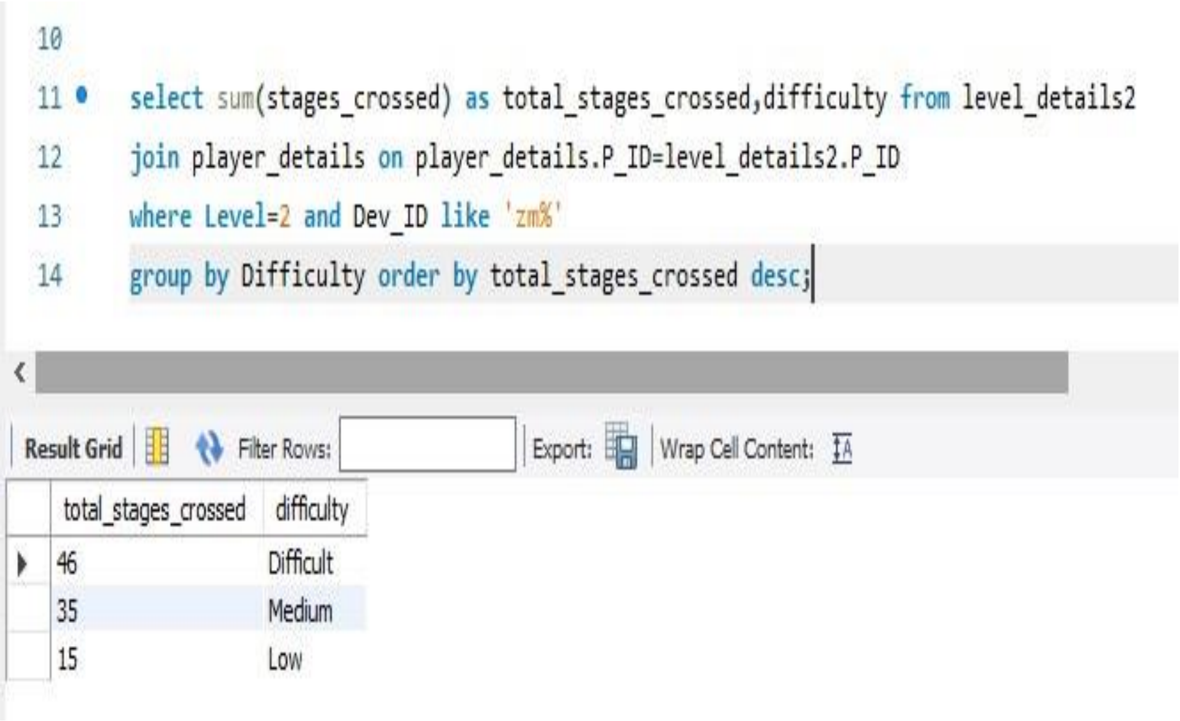
l1_code	avg_kill_count
war_zone	19.2857
bulls_eye	22.2500
speed_blitz	19.3333

Query3:

Find the total number of stages crossed at each difficulty level where for Level2 with players use zm_series device. Arrange the result.

select sum(stages_crossed) as total_stages_crossed,difficulty from level_details2 join player_details on player_details.P_ID=level_details2.P_ID where Level=2 and Dev_ID like 'zm%' group by Difficulty order by total_stages_crossed desc;

Analysis – it performs an inner join on with Player_Details table based on the Player Id. The result is grouped by difficulty and ordered by the



```
select sum(stages_crossed) as total_stages_crossed,difficulty from level_details2
join player_details on player_details.P_ID=level_details2.P_ID
where Level=2 and Dev_ID like 'zm%'
group by Difficulty order by total_stages_crossed desc;
```

total_stages_crossed	difficulty
46	Difficult
35	Medium
15	Low

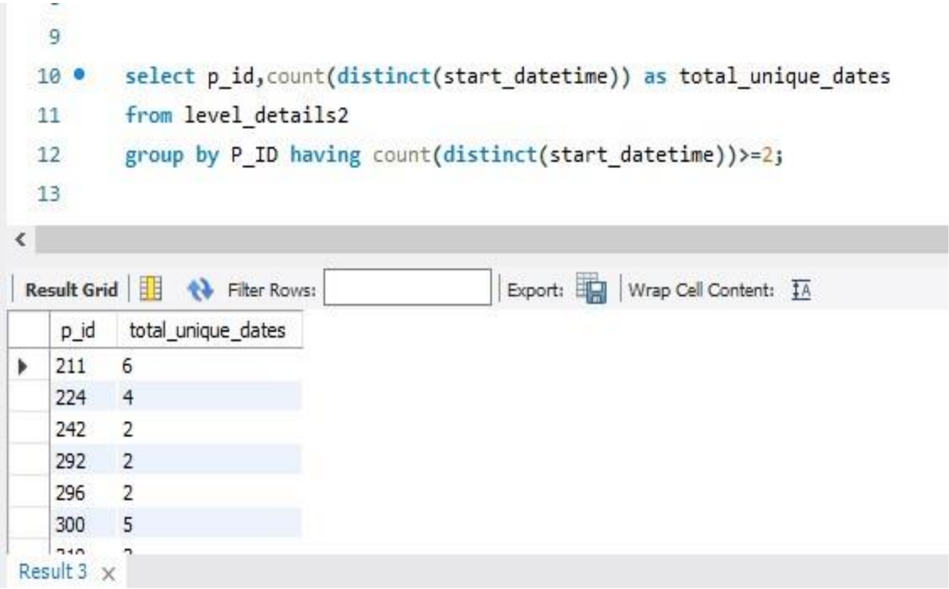
Query 4 :

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(start_datetime)) astotal_unique_dates from level_details2

group by P_ID having

count(distinct(start_datetime))>=2;
```



```
select p_id,count(distinct(start_datetime)) as total_unique_dates
from level_details2
group by P_ID having count(distinct(start_datetime))>=2;
```

p_id	total_unique_dates
211	6
224	4
242	2
292	2
296	2
300	5

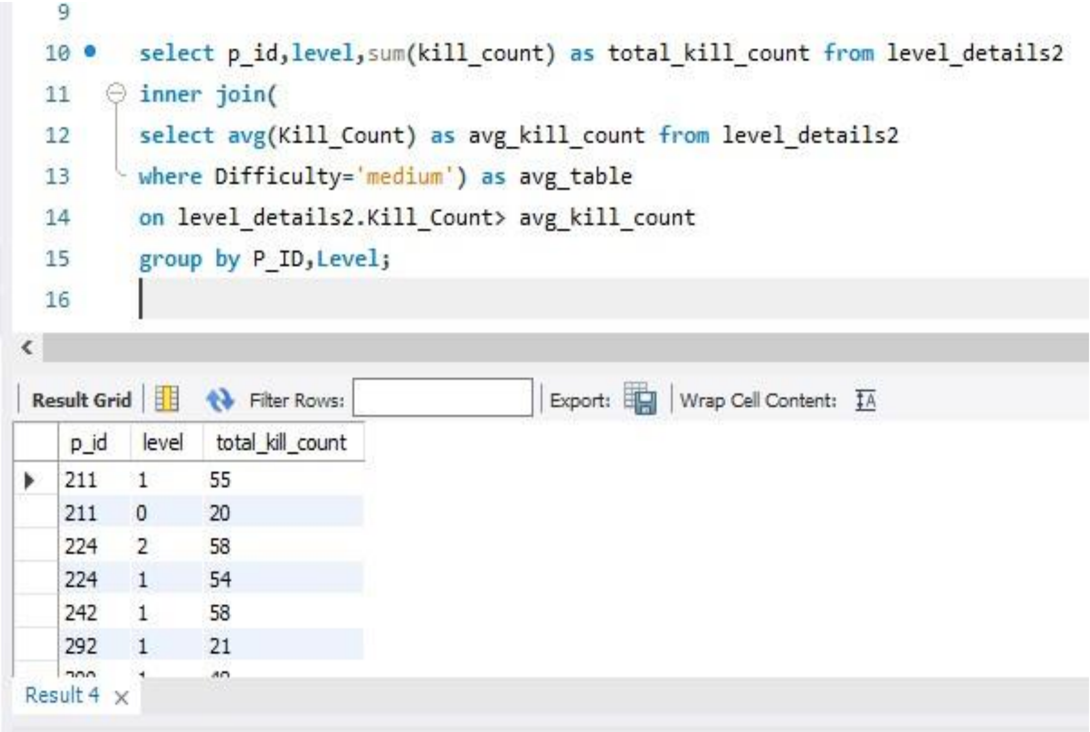
Analysis – It groups the results by p_ID and filter out the groups where the count of unique dates is greater than 1. This query helps identify players who have started games on multiple dates

Query 5:

Find P_ID and level wise sum of kill_counts where kill_count is greater than avg kill count for the Medium difficulty

```
select p_id,level,sum(kill_count) as total_kill_count from level_details2 inner
join( select avg(Kill_Count) as avg_kill_count from level_details2 where
Difficulty='medium') as avg_table on level_details2.Kill_Count> avg_kill_count
group by P_ID,Level;
```

Analysis – It filters the data based on the conditions that the Kill_Count is greater than the average Kill_Count for records with the Difficulty_level set to ‘Medium’. Finally,it groups the result by P_ID and Level. This query helps identify players who have achieved above-average kill



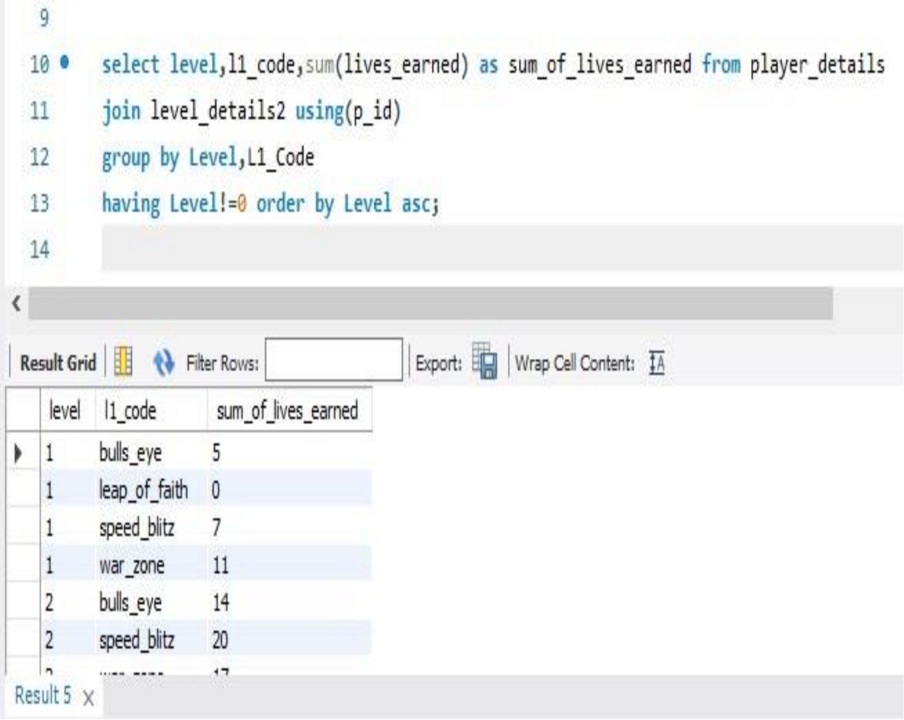
```
select p_id,level,sum(kill_count) as total_kill_count from level_details2
inner join(
select avg(Kill_Count) as avg_kill_count from level_details2
where Difficulty='medium') as avg_table
on level_details2.Kill_Count> avg_kill_count
group by P_ID,Level;
```

p_id	level	total_kill_count
211	1	55
211	0	20
224	2	58
224	1	54
242	1	58
292	1	21

Query 6:

Find Level and its corresponding Level code wise sum of lives earned excluding level 0. Arrange in ascending order of level.

```
select level,l1_code,sum(lives_earned) as sum_of_lives_earned from player_details join level_details2 using(p_id) group by Level,L1_Code having Level!=0 order by Level asc;
```



```
select level,l1_code,sum(lives_earned) as sum_of_lives_earned from player_details
join level_details2 using(p_id)
group by Level,L1_Code
having Level!=0 order by Level asc;
```

level	l1_code	sum_of_lives_earned
1	bulls_eye	5
1	leap_of_faith	0
1	speed_blitz	7
1	war_zone	11
2	bulls_eye	14
2	speed_blitz	20

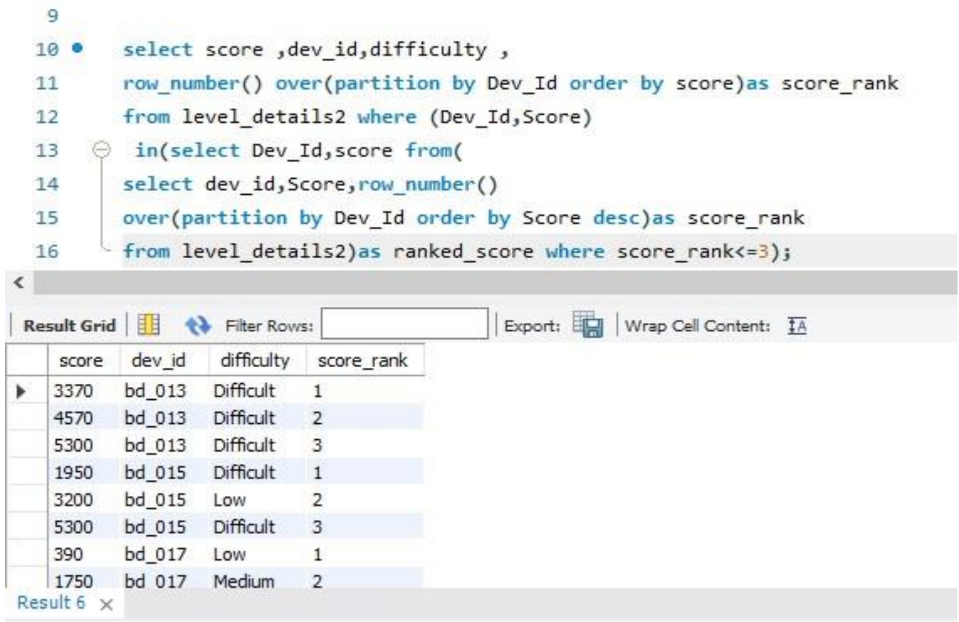
Analysis – It filters the data to exclude Level 0. which typically represents the initial level or setup phase. Then, it calculates the sum of lives earned for each level and groups the results by Level and Level_code. Finally, it orders the results by Level in ascending order.

Query 7:

Find Top 3 score based on each dev_id and Rank them in increasing order using Row_Number. Display difficulty as well.

```
    select score ,dev_id,difficulty , row_number() over(partition by Dev_Id order by score)as score_rank
from level_details2 where (Dev_Id,Score) in(select Dev_Id,score from( select
dev_id,Score,row_number() over(partition by Dev_Id order by Score desc)as score_rank from
level_details2)as ranked_score where score_rank<=3);
```

Analysis – By partitioning the data by Developer_ID and ranking scores within each group, the query efficiently retrieves the highest scores. The main query then selects the Developer_ID. Difficulty Level,Score,and Rank



```
select score ,dev_id,difficulty ,
row_number() over(partition by Dev_Id order by score)as score_rank
from level_details2 where (Dev_Id,Score)
 in(select Dev_Id,score from(
select dev_id,Score,row_number()
over(partition by Dev_Id order by Score desc)as score_rank
from level_details2)as ranked_score where score_rank<=3);
```

score	dev_id	difficulty	score_rank
3370	bd_013	Difficult	1
4570	bd_013	Difficult	2
5300	bd_013	Difficult	3
1950	bd_015	Difficult	1
3200	bd_015	Low	2
5300	bd_015	Difficult	3
390	bd_017	Low	1
1750	bd_017	Medium	2

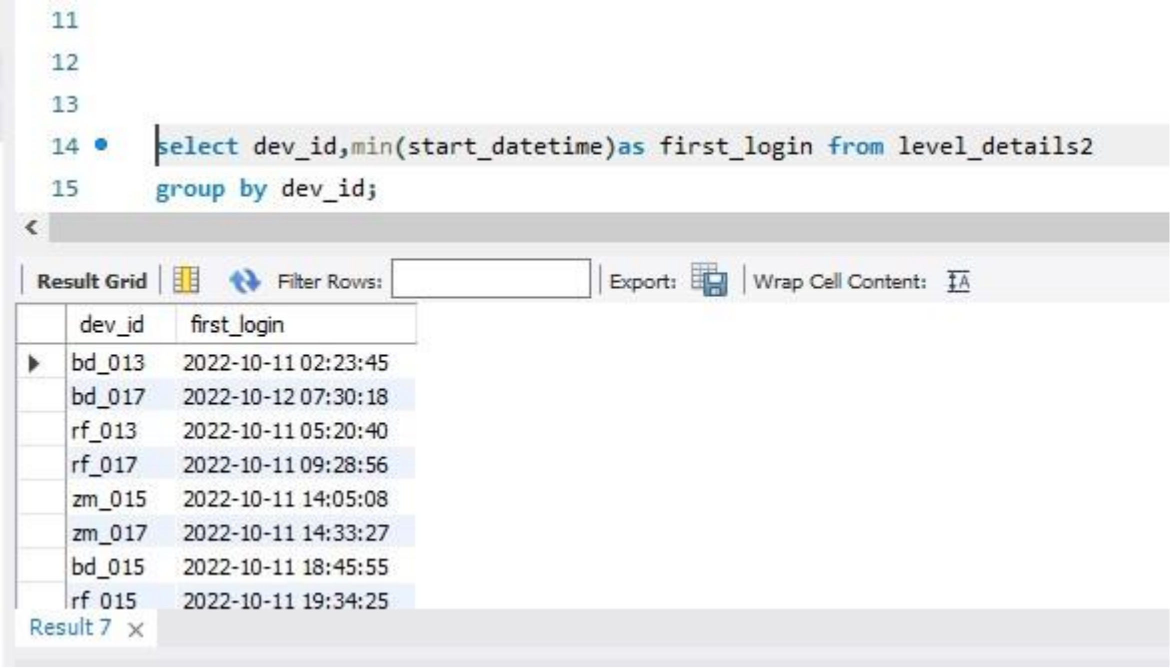
Query8:

Final first_login datetime for each device id.

```
select dev_id,min(start_datetime)as first_login from level_details2

group by dev_id;
```

Analysis – The SQL query retrieves the earliest login timestamp for each developer by selecting the minimum start datetime grouped by the developer’s ID from the Player_Details.



```
select dev_id,min(start_datetime)as first_login from level_details2
group by dev_id;
```

dev_id	first_login
bd_013	2022-10-11 02:23:45
bd_017	2022-10-12 07:30:18
rf_013	2022-10-11 05:20:40
rf_017	2022-10-11 09:28:56
zm_015	2022-10-11 14:05:08
zm_017	2022-10-11 14:33:27
bd_015	2022-10-11 18:45:55
rf_015	2022-10-11 19:34:25

9

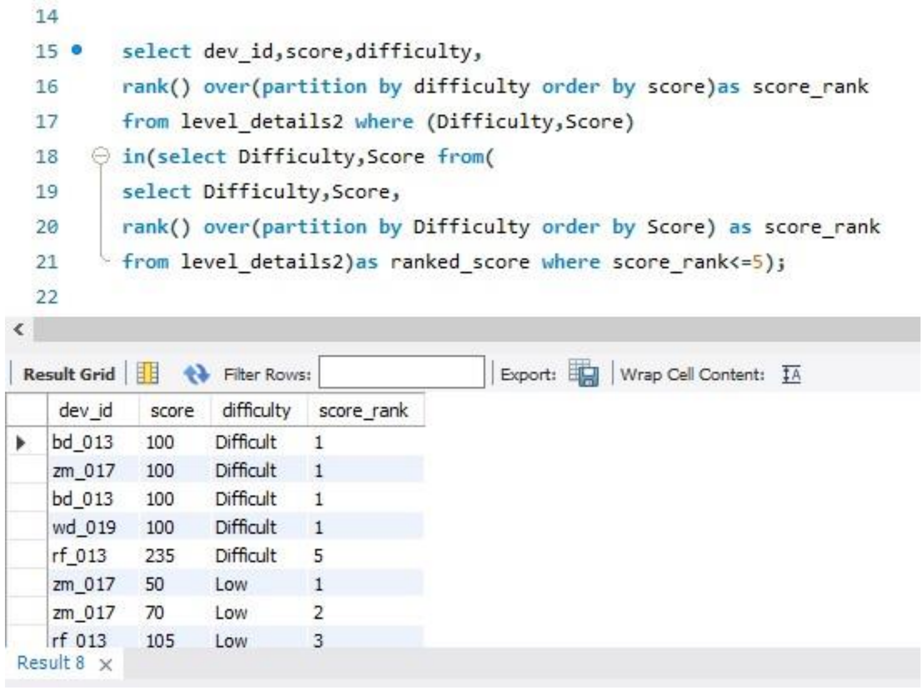
Query 9:

Find Top 5 Score based on each difficulty level and Rank them in increasing order using Rank.

Display dev_id as well.

```
select dev_id,score,difficulty,rank()over(partition by
difficulty order by score)as score_rank from level_details2 where
(Difficulty,Score) in(select Difficulty,Score
from(select Difficulty,Score rank() over(partition by
Difficulty order by Score) as score_rank
from
level_details2)as ranked_score where score_rank<=5);
```

Analysis – It assigns a rank to each score based on descending order. Then, it selects the developer ID, difficulty level, score and rank from the



```
select dev_id,score,difficulty,
rank() over(partition by difficulty order by score)as score_rank
from level_details2 where (Difficulty,Score)
in(select Difficulty,Score from(
select Difficulty,Score,
rank() over(partition by Difficulty order by Score) as score_rank
from level_details2)as ranked_score where score_rank<=5);
```

dev_id	score	difficulty	score_rank
bd_013	100	Difficult	1
zm_017	100	Difficult	1
bd_013	100	Difficult	1
wd_019	100	Difficult	1
rf_013	235	Difficult	5
zm_017	50	Low	1
zm_017	70	Low	2
rf_013	105	Low	3

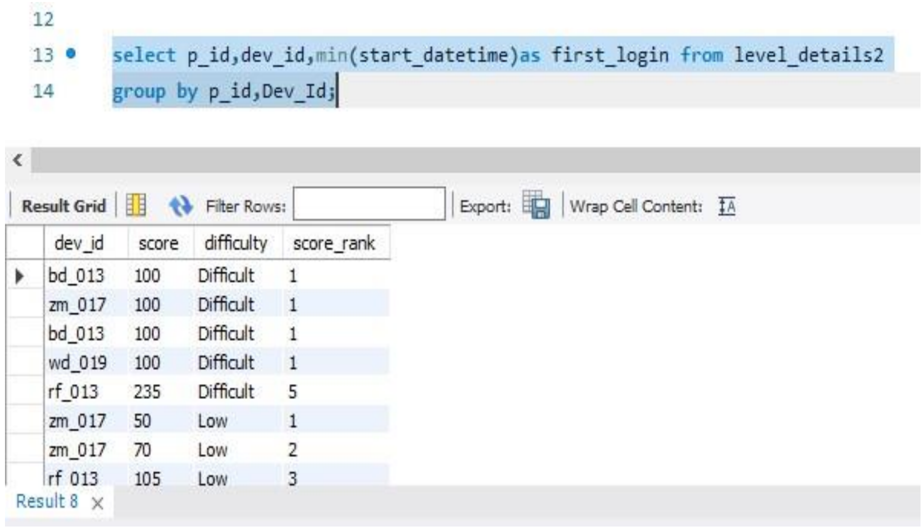
TopScores CTE where the rank is less than or equal to 5.

Query10:

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(start_datetime)as first_login from level_details2 group by p_id,Dev_Id;
```

Analysis – It assigns a row number to each login record within each player’s data, ordered by the start_datetime. Then, it selects the player ID(P_ID),developer ID(Dev_ID), and the start_datetime corresponding to the first login



```
select p_id,dev_id,min(start_datetime)as first_login from level_details2
group by p_id,Dev_Id;
```

dev_id	score	difficulty	score_rank
bd_013	100	Difficult	1
zm_017	100	Difficult	1
bd_013	100	Difficult	1
wd_019	100	Difficult	1
rf_013	235	Difficult	5
zm_017	50	Low	1
zm_017	70	Low	2
rf 013	105	Low	3

(identified by RowNum = 1) from the FirstLogin CTE.

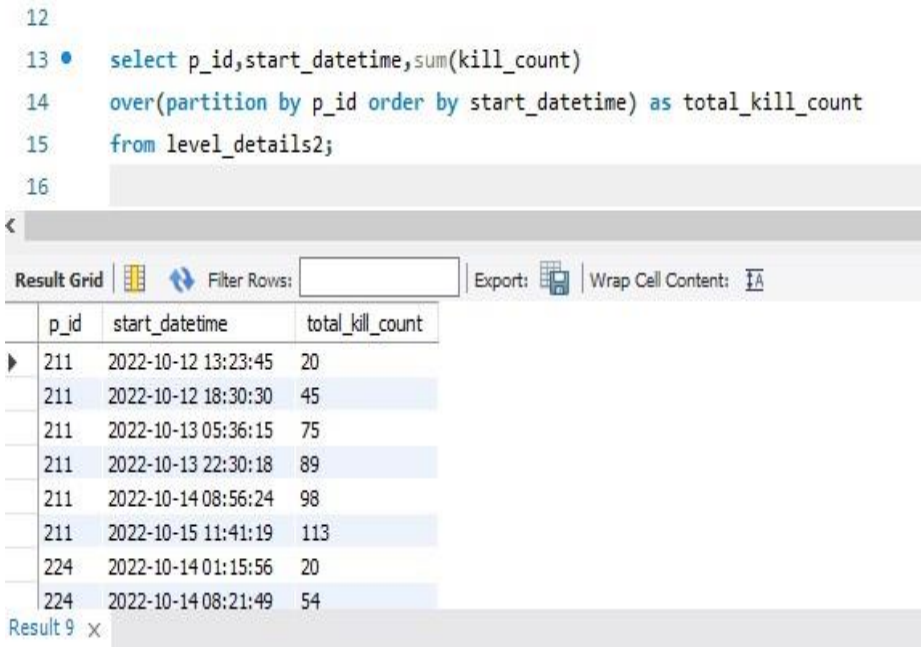
Query11:

For each player and date, how many kill_count played so far by the player. That is, the total number of games played by the player until that date.

A. Window function

```
Select
p_id,start_datetime,sum(kill
_co un over(partition by
p_id order
by
start_datetime)
as
total_kill_count
from level_details2;
```

Analysis – It utilizes the window function SUM() with the OVER() clause to partition the data by P_ID and order it by start_datetime. This allows tracing the total kill count accumulated by each player as they progress through the



```
select p_id,start_datetime,sum(kill_count)
over(partition by p_id order by start_datetime) as total_kill_count
from level_details2;
```

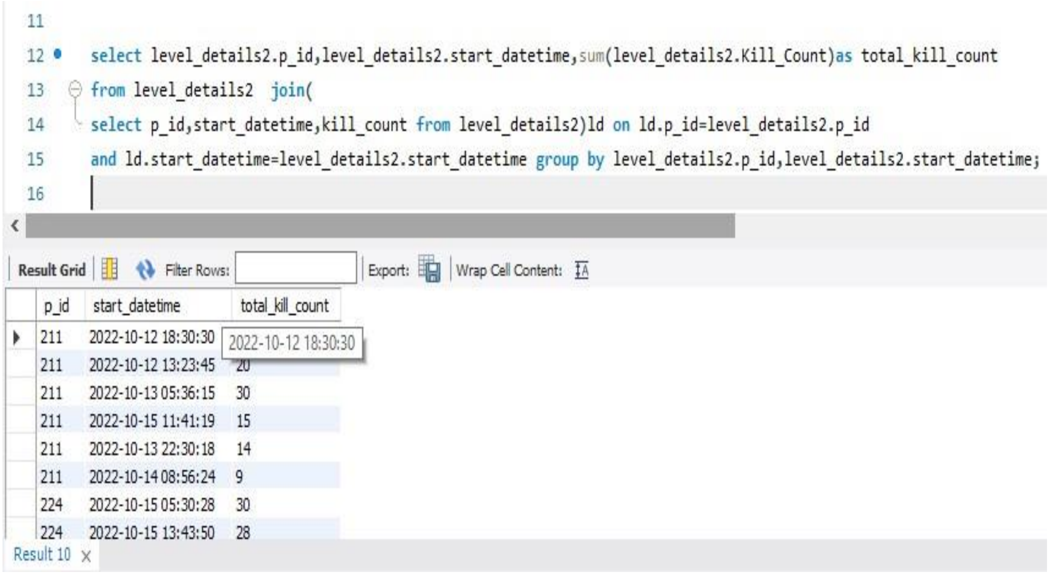
p_id	start_datetime	total_kill_count
211	2022-10-12 13:23:45	20
211	2022-10-12 18:30:30	45
211	2022-10-13 05:36:15	75
211	2022-10-13 22:30:18	89
211	2022-10-14 08:56:24	98
211	2022-10-15 11:41:19	113
224	2022-10-14 01:15:56	20
224	2022-10-14 08:21:49	54

game sessions, adding in analyzing player performance trends and engagement levels over time.

B.Without Window Function

```
Select
level_details2.p_id,level_details2.start_datetime,sum(lev
el_details2.Kill_Count)as  total_kill_count              from
level_details2  join(
select   p_id,start_datetime,kill_count   from
level_details2)ld on ld.p_id=level_details2.p_id and
ld.start_datetime=level_details2.start_datetime group by
level_details2.p_id,level_details2.start_datetime;
```

Analysis – It utilizes a correlated subquery to sum the Kill_count values from the Game_Data table for each player where the start_Datetime is less than or equal to the start_datetime of the current row. This provides a running total of kill counts for each player as they progess through their gaming session



```
select level_details2.p_id,level_details2.start_datetime,sum(level_details2.Kill_Count)as total_kill_count
from level_details2  join(
select p_id,start_datetime,kill_count from level_details2)ld on ld.p_id=level_details2.p_id
and ld.start_datetime=level_details2.start_datetime group by level_details2.p_id,level_details2.start_datetime;
```

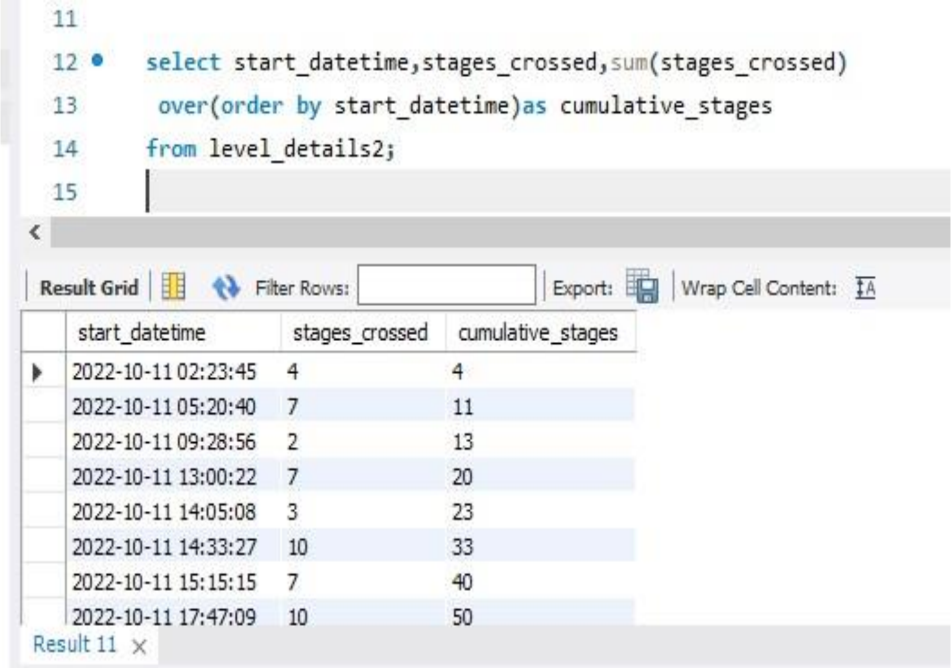
p_id	start_datetime	total_kill_count
211	2022-10-12 18:30:30	[illegible]
211	2022-10-12 13:23:45	20
211	2022-10-13 05:36:15	30
211	2022-10-15 11:41:19	15
211	2022-10-13 22:30:18	14
211	2022-10-14 08:56:24	9
224	2022-10-15 05:30:28	30
224	2022-10-15 13:43:50	28

Query12:

Find the cumulative sum of stages crossed over a start_datetime for each player id but exclude the most recent start_datetime

```
select
start_datetime,stages_crossed,sum(stages_crossed)
over(order by start_datetime)as cumulative_stages from
level_details2;
```

Analysis – It utilizes the SUM() function with the window function OVER() to sum the stages_crossed values from the Game Data Table for each player. THE ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING clause specifies the range of rows to include in the sum, which in this case is from the beginning of the partition (UNBOUNDED PRECEDING) up to the row immediately preceding the current row.



```
select start_datetime,stages_crossed,sum(stages_crossed)
 over(order by start_datetime)as cumulative_stages
from level_details2;
```

start_datetime	stages_crossed	cumulative_stages
2022-10-11 02:23:45	4	4
2022-10-11 05:20:40	7	11
2022-10-11 09:28:56	2	13
2022-10-11 13:00:22	7	20
2022-10-11 14:05:08	3	23
2022-10-11 14:33:27	10	33
2022-10-11 15:15:15	7	40
2022-10-11 17:47:09	10	50

Query13:

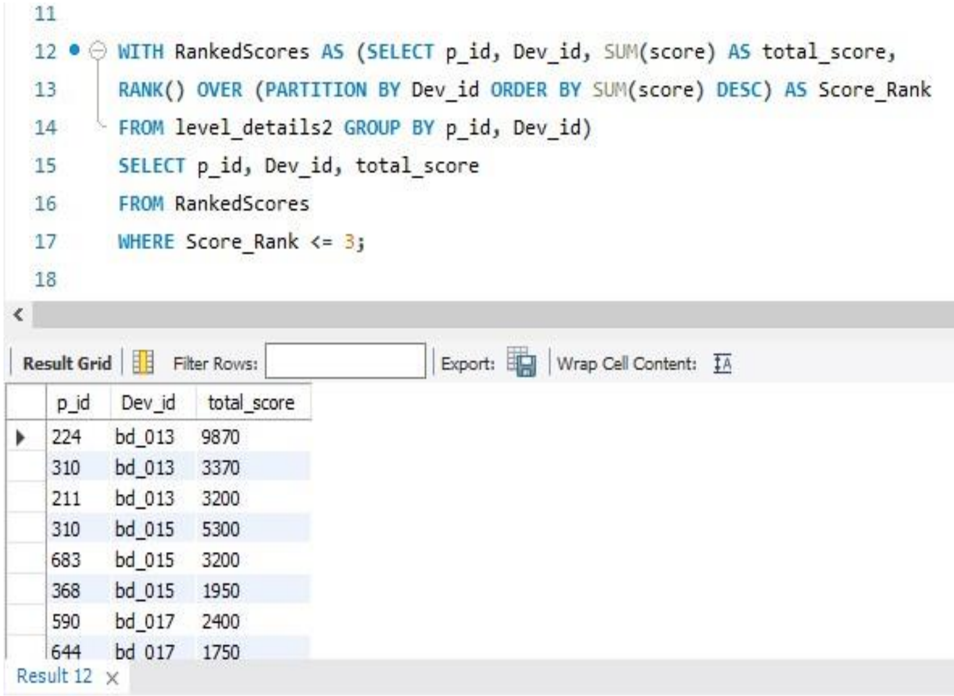
Extract top 3 highest sum of score for each device id and the corresponding player_id

WITH RankedScores AS (SELECT p_id, Dev_id, SUM(score) AS total_score, RANK() OVER (PARTITION BY Dev_id ORDER BY SUM(score) DESC) AS Score_Rank FROM level_details2 GROUP BY p_id, Dev_id) SELECT p_id, Dev_id, total_score

FROM RankedScores WHERE Score_Rank <= 3;

Analysis – It then assigns a rank to each player within each device based on their total score, with the highest scorer receiving rank1.

The results are filtered to include only the top 3 scorers for each device, showing their P_ID, Dev_ID and total_score.



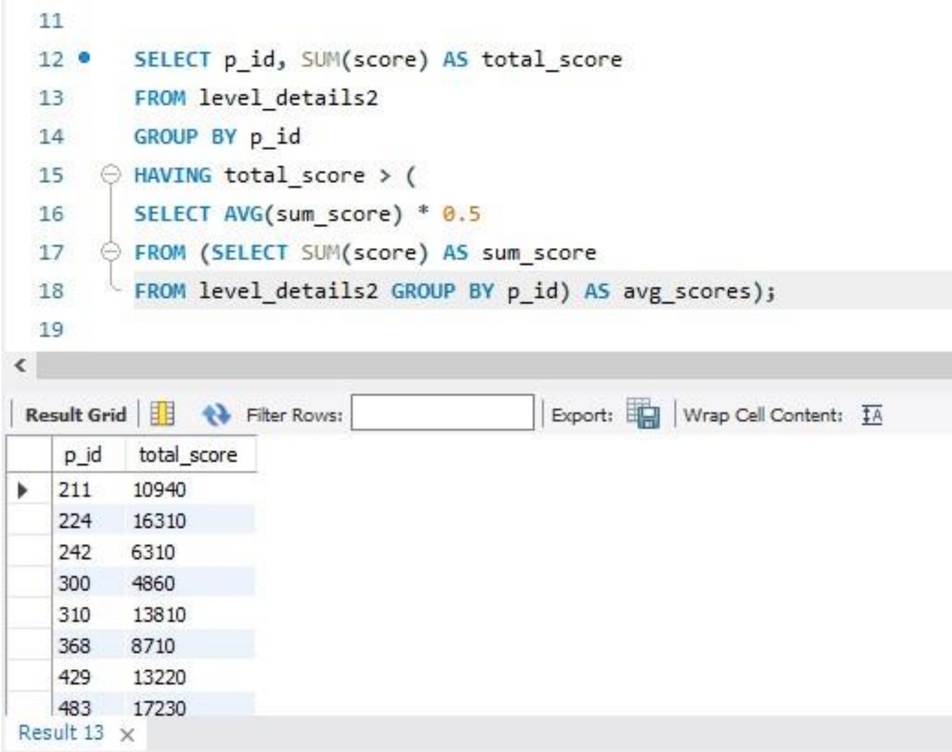
```
WITH RankedScores AS (SELECT p_id, Dev_id, SUM(score) AS total_score,
RANK() OVER (PARTITION BY Dev_id ORDER BY SUM(score) DESC) AS Score_Rank
FROM level_details2 GROUP BY p_id, Dev_id)
SELECT p_id, Dev_id, total_score
FROM RankedScores
WHERE Score_Rank <= 3;
```

p_id	Dev_id	total_score
224	bd_013	9870
310	bd_013	3370
211	bd_013	3200
310	bd_015	5300
683	bd_015	3200
368	bd_015	1950
590	bd_017	2400
644	bd 017	1750

Query14:

Find players who scored more than 50% of the avg score scored by sum of scores for each player_id.

```
SELECT p_id, SUM(score) AS total_score FROM level_details2
GROUP BY p_id HAVING total_score > (SELECT AVG(sum_score)
* 0.5
        FROM (SELECT SUM(score) AS sum_score   FROM level_details2
        GROUP BY p_id) AS avg_scores);
```



```
SELECT p_id, SUM(score) AS total_score
FROM level_details2
GROUP BY p_id
HAVING total_score > (
SELECT AVG(sum_score) * 0.5
FROM (SELECT SUM(score) AS sum_score
FROM level_details2 GROUP BY p_id) AS avg_scores);
```

p_id	total_score
211	10940
224	16310
242	6310
300	4860
310	13810
368	8710
429	13220
483	17230

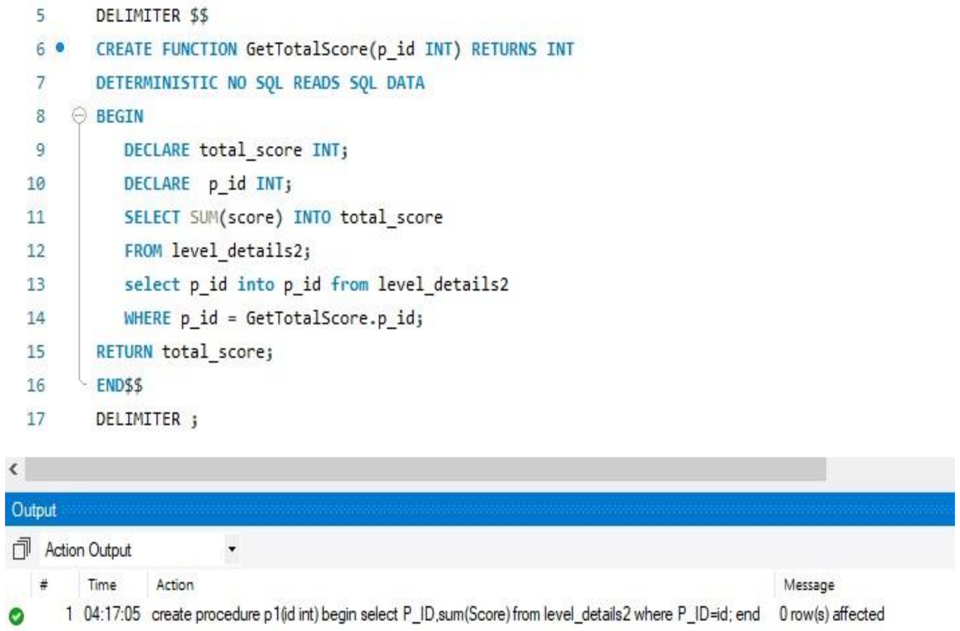
Analysis – It calculates the total score for each player In the inner subquery and then filters the results based on the condition specified.

Query15:

Create a function to return sum of score for a given player_id

```
DELIMITER $$
CREATE FUNCTION GetTotalScore(p_id INT) RETURNS INT DETERMINISTIC NO SQL
READS SQL DATA
    BEGIN
      DECLARE total_score INT;
      DECLARE  p_id INT;
      SELECT SUM(score) INTO total_score    FROM level_details2;   select p_id
    into p_id from level_details2  WHERE p_id = GetTotalScore.p_id;
      RETURN total_score;
    END$$
DELIMITER ;
```

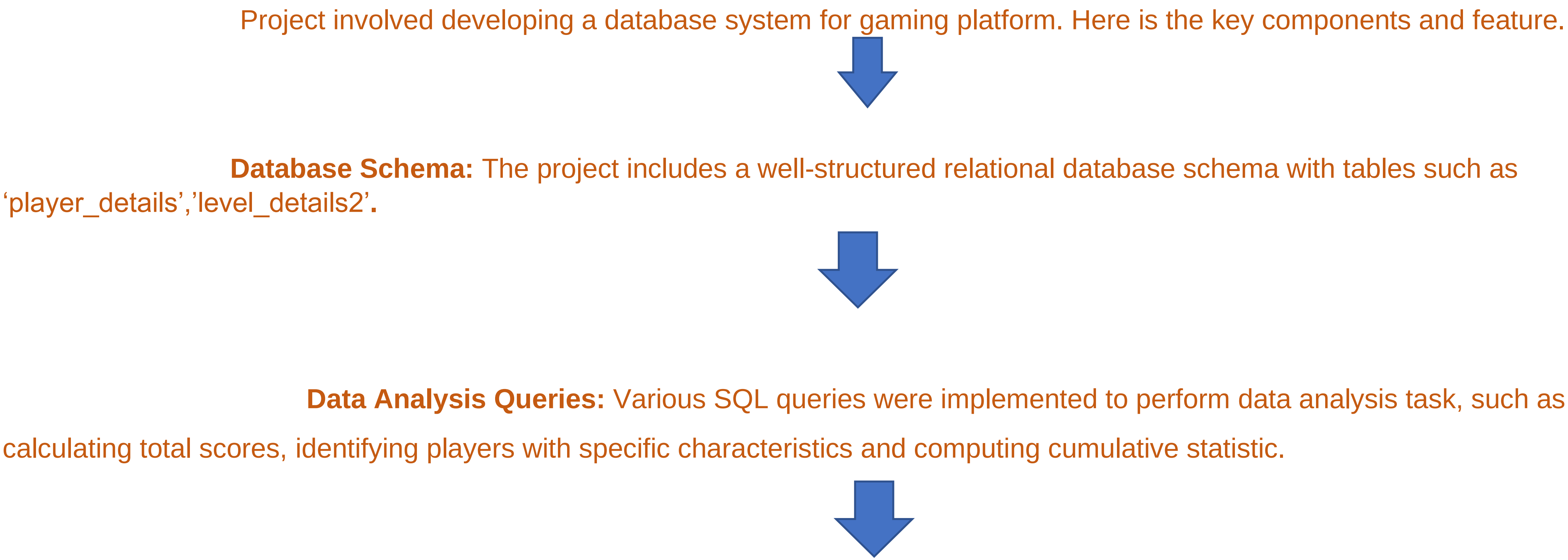
Analysis – returns the sum of scores for that player from the LEVEL_DETAILS table. After creating the function, it selects and



```
DELIMITER $$
CREATE FUNCTION GetTotalScore(p_id INT) RETURNS INT
DETERMINISTIC NO SQL READS SQL DATA
BEGIN
   DECLARE total_score INT;
   DECLARE  p_id INT;
   SELECT SUM(score) INTO total_score
   FROM level_details2;
   select p_id into p_id from level_details2
   WHERE p_id = GetTotalScore.p_id;
RETURN total_score;
END$$
DELIMITER ;
```

```
   #   Time      Action                                                                       Message
✓  1   04:17:05  create procedure p1(id int) begin select P_ID,sum(Score) from level_details2 where P_ID=id; end   0 row(s) affected
```

Summary



Project involved developing a database system for gaming platform. Here is the key components and feature.

Database Schema: The project includes a well-structured relational database schema with tables such as ‘player_details’,’level_details2’**.**

Data Analysis Queries: Various SQL queries were implemented to perform data analysis task, such as calculating total scores, identifying players with specific characteristics and computing cumulative statistic.

Stored procedures and functions: MY SQL stored procedures and function were utilized to encapsulate complex SQL logic, improve code modularity and enhance Database performance.

Windows Function: Windows function such as ‘ROW_NUMBER(), OVER(), SUM(), were designed to perform advanced analytical skills operations like ranking scores, cumulative sum and retrieving data based on specific partitions.

Optimized Queries: Effort were made to optimize SQL queries for efficiency and performance, ensuring that data retrieval and processing task are executed swiftly, even with the large set.

Overall the project demonstrates proficiency in database design, SQL Programming and data analysis techniques. Providing valuable insight into player behaviour and game performance matrics.