Decode Gaming Behavour

Amalai Rham Machunga

CONTENTS

Objective

During this project, I analyzed gaming data to explore gaming behavior, which involved working with two tables: 'Player Details' and 'Level Details'.

Dataset Description

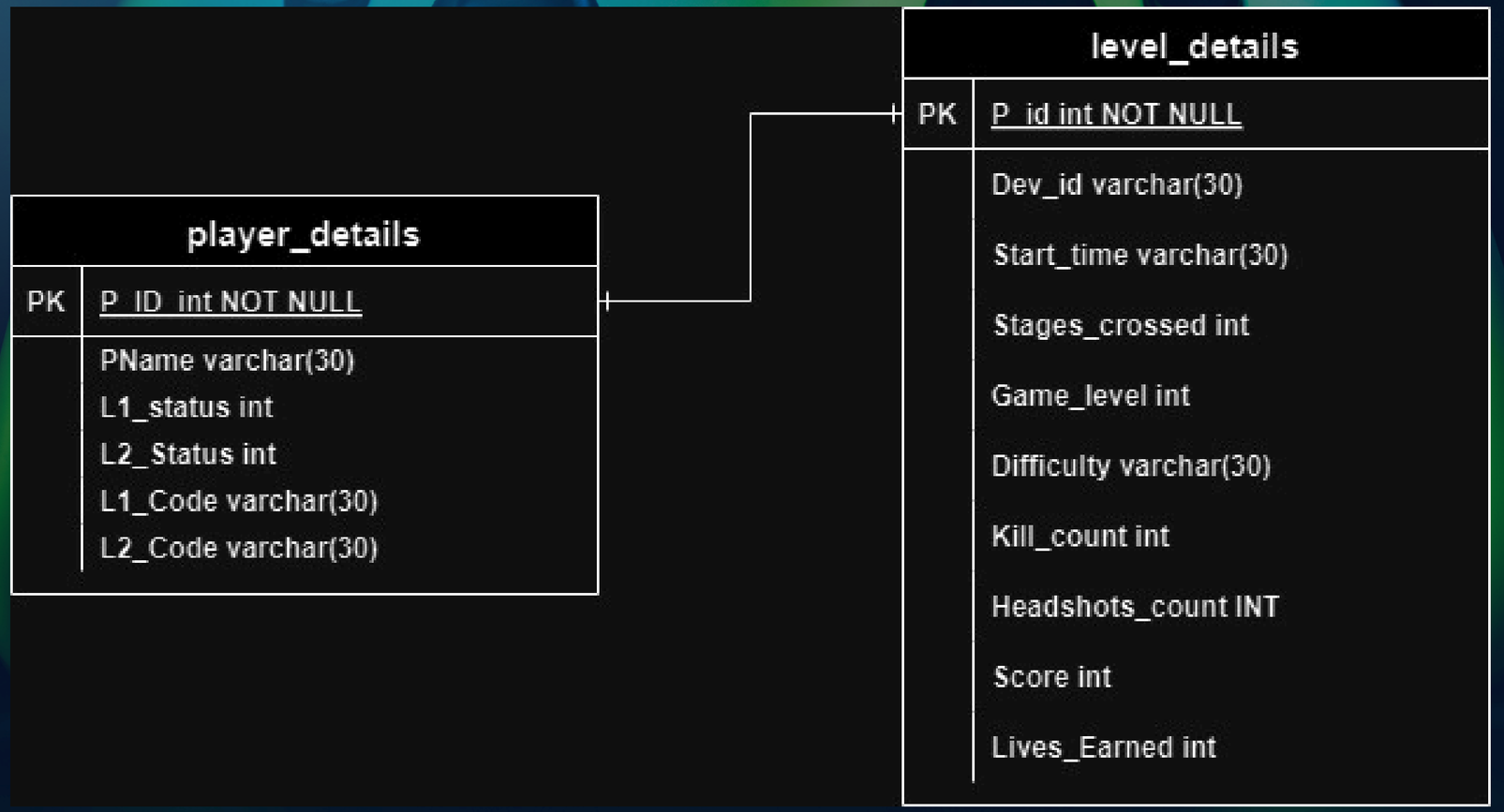
Player Details Table:

- `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

Entity Relationship Diagram



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

level_details

Limit to 1000 rows

```
1    SELECT level_details.P_ID, level_details.Dev_ID, player_details.PName, level_details.Di
2    FROM game_analysis.level_details
3    JOIN game_analysis.player_details
4    ON level_details.P_ID = player_details.P_ID
5    WHERE Level = 0;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

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

```
SELECT player_details.L1_code, ROUND(AVG(level_details.Kill_Count),2) AS Kill_Count
FROM player_details
LEFT JOIN level_details
ON level_details.P_ID = player_details.P_ID
WHERE Lives_Earned = 2 AND Stages_crossed >=3
GROUP BY L1_code;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

L1_code	Kill_Count
speed_blitz	19.33
war_zone	19.29
bulls_eye	22.25

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.

```
1  SELECT SUM(level_details.Stages_crossed) AS Total_Num_of_Stages_Crossed , Dev_ID, level_details.Difficulty
2  FROM level_details
3  WHERE Level = 2 AND Dev_ID LIKE 'zm%'
4  GROUP BY Difficulty, DEV_ID
5  ORDER BY Total_Num_of_Stages_Crossed DESC;
6
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Total_Num_of_Stages_Crossed	Dev_ID	Difficulty
39	zm_017	Difficult
21	zm_017	Medium
14	zm_015	Medium
10	zm_017	Low
7	zm_013	Difficult
5	zm_015	Low

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

```
SELECT level_details.P_ID, COUNT(DISTINCT level_details.Start_time) AS Total_Num_of_Unique_Dates
FROM level_details
GROUP BY P_ID
HAVING COUNT(DISTINCT level_details.Start_time) > 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

P_ID	Total_Num_of_Unique_Dates
211	6
224	4
242	2
292	2
296	2
300	5
310	3
358	2
368	4
429	4
483	5
547	3
590	5
632	5
641	3
644	3
656	4
663	5
683	7

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

```
1  SELECT level_details.P_ID, Level, SUM(level_details.Kill_Count) AS Total_Kill_Count
2  FROM level_details
3  WHERE Kill_Count > ( SELECT AVG(level_details.Kill_Count) AS Average_Kill_Count
4  FROM level_details
5  WHERE Difficulty = 'Medium')
6  GROUP BY P_ID, Level
7  ORDER BY P_ID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

P_ID	Level	Total_Kill_Count
211	0	20
211	1	55
224	1	54
224	2	58
242	1	58
292	1	21
300	1	48
310	0	34
310	1	20
368	1	20
368	2	24
429	1	30
429	2	55
483	1	40
483	2	94
547	1	20
558	0	21
590	1	24
632	0	45
632	1	28
632	2	53
644	2	24
656	1	37

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

```sql
SELECT level_details.Level, COALESCE(player_details.L1_code, player_details.L2_code) AS Level_Code,
       SUM(level_details.Lives_Earned) AS Total_Lives_Earned
FROM level_details
LEFT JOIN player_details
ON level_details.P_ID = player_details.P_ID
WHERE Level != 0
GROUP BY Level, L1_code, L2_code
ORDER BY Level;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Level	Level_Code	Total_Lives_Earned
1	bulls_eye	3
1	bulls_eye	1
1	bulls_eye	1
1	leap_of_faith	0
1	speed_blitz	0
1	speed_blitz	4
1	speed_blitz	3
1	war_zone	4
1	war_zone	0
1	war_zone	7
2	bulls_eye	6
2	bulls_eye	8
2	speed_blitz	6
2	speed_blitz	14
2	war_zone	3
2	war_zone	14

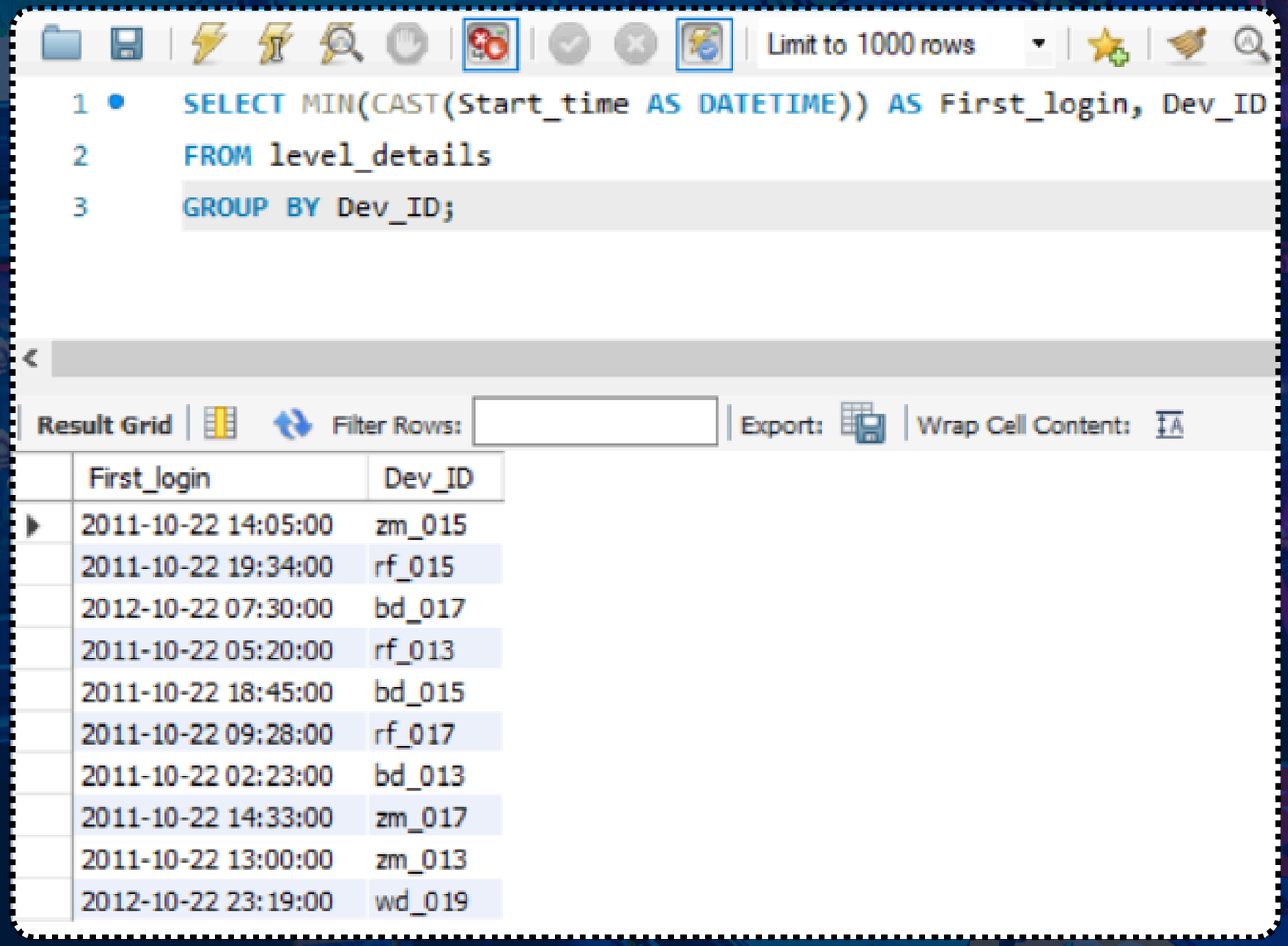
7. Find the top 3 scores based on each `Dev_ID` and rank them in increasing order using `Row_Number`. Display the difficulty as well.

```sql
WITH Top_Scores AS (
    SELECT Score, Dev_ID, Difficulty, ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY Score DESC) AS Ranking
    FROM level_details
)
SELECT Score, Dev_ID, Difficulty, Ranking
FROM Top_Scores
WHERE Ranking <= 3;
```

Result Grid

Score	Dev_ID	Difficulty	Ranking
5300	bd_013	Difficult	1
4570	bd_013	Difficult	2
3370	bd_013	Difficult	3
5300	bd_015	Difficult	1
3200	bd_015	Low	2
1950	bd_015	Difficult	3
2400	bd_017	Low	1
1750	bd_017	Medium	2
390	bd_017	Low	3
2970	rf_013	Difficult	1
2700	rf_013	Medium	2
2300	rf_013	Medium	3
3950	rf_015	Difficult	1
2800	rf_015	Medium	2
900	rf_015	Medium	3
5140	rf_017	Difficult	1
5140	rf_017	Medium	2
3500	rf_017	Difficult	3
4390	wd_019	Difficult	1
1550	wd_019	Low	2
635	wd_019	Difficult	3
4710	zm_013	Difficult	1
2350	zm_013	Medium	2

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



```
Limit to 1000 rows

1  SELECT MIN(CAST(Start_time AS DATETIME)) AS First_login, Dev_ID
2  FROM level_details
3  GROUP BY Dev_ID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

First_login	Dev_ID
2011-10-22 14:05:00	zm_015
2011-10-22 19:34:00	rf_015
2012-10-22 07:30:00	bd_017
2011-10-22 05:20:00	rf_013
2011-10-22 18:45:00	bd_015
2011-10-22 09:28:00	rf_017
2011-10-22 02:23:00	bd_013
2011-10-22 14:33:00	zm_017
2011-10-22 13:00:00	zm_013
2012-10-22 23:19:00	wd_019

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

```sql
WITH Top_Scores AS(
    SELECT Dev_ID, Score, Difficulty, RANK() OVER (PARTITION BY Difficulty ORDER BY Score DESC) AS Ranking
    FROM level_details
)
SELECT Dev_ID, Score, Difficulty, Ranking
FROM Top_Scores
WHERE Ranking <=5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Dev_ID	Score	Difficulty	Ranking
zm_017	5500	Difficult	1
zm_017	5500	Difficult	1
bd_015	5300	Difficult	3
bd_013	5300	Difficult	3
rf_017	5140	Difficult	5
zm_015	3470	Low	1
zm_017	3210	Low	2
bd_015	3200	Low	3
bd_013	2840	Low	4
zm_015	2800	Low	5
zm_017	5490	Medium	1
rf_017	5140	Medium	2
zm_015	4950	Medium	3
zm_015	4950	Medium	3
rf_015	2800	Medium	5

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.

```
1 •  SELECT P_ID, Dev_ID, MIN(CAST(Start_time AS DATETIME)) AS First_Login
2    FROM level_details
3    GROUP BY P_ID, Dev_ID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

P_ID	Dev_ID	First_Login
483	wd_019	2013-10-22 06:20:00
368	zm_015	2012-10-22 01:14:00
368	zm_017	2012-10-22 04:20:00
368	bd_015	2012-10-22 11:59:00
368	rf_013	2015-10-22 14:47:00
663	bd_013	2015-10-22 17:30:00
663	rf_013	2015-10-22 19:36:00
663	wd_019	2015-10-22 06:30:00
663	zm_015	2015-10-22 09:56:00
663	zm_017	2015-10-22 23:41:00
683	bd_013	2011-10-22 02:23:00
683	bd_015	2011-10-22 18:45:00
683	rf_013	2012-10-22 14:36:00
683	rf_015	2013-10-22 22:30:00
683	zm_017	2013-10-22 08:16:00
358	zm_017	2014-10-22 05:05:00
358	zm_013	2014-10-22 18:23:00
641	rf_013	2014-10-22 01:25:00
641	rf_015	2013-10-22 04:04:00
558	wd_019	2012-10-22 23:19:00

11. For each player and date, determine how many `kill_counts` were played by the player so far.

a) Using Window Functions

```
1    # With Window Function
2 •  SELECT P_ID, DATE(Start_time) AS DATE, SUM(Kill_Count) OVER (PARTITION BY P_ID ORDER BY Start_time) AS Total_Kill_Counts
3    FROM level_details;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

P_ID	DATE	Total_Kill_Counts
211	2012-10-22	20
211	2012-10-22	45
211	2013-10-22	59
211	2013-10-22	89
211	2014-10-22	98
211	2015-10-22	113
224	2014-10-22	20
224	2014-10-22	54
224	2015-10-22	82
224	2015-10-22	112
242	2013-10-22	21
242	2014-10-22	58
292	2012-10-22	21
292	2015-10-22	25
296	2014-10-22	7
296	2014-10-22	11
300	2011-10-22	25
300	2011-10-22	48
300	2012-10-22	52
300	2012-10-22	66
300	2013-10-22	74
310	2011-10-22	20
310	2013-10-22	54
310	2015-10-22	68

b) Without Window Functions

```
1    # Without Window Function
2 •  SELECT P_ID, DATE(Start_time) AS Date, SUM(Kill_Count) AS Total_Kill_Counts
3    FROM level_details
4    GROUP BY P_ID, Date;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

P_ID	Date	Total_Kill_Counts
644	2011-10-22	18
644	2012-10-22	24
656	2015-10-22	15
656	2013-10-22	19
656	2014-10-22	3
656	2011-10-22	18
296	2014-10-22	11
632	2012-10-22	73
632	2013-10-22	27
632	2014-10-22	30
428	2015-10-22	5
429	2011-10-22	99
310	2011-10-22	20
310	2013-10-22	34
310	2015-10-22	14
211	2012-10-22	45
211	2013-10-22	44
211	2014-10-22	9
211	2015-10-22	15
319	2012-10-22	5
547	2015-10-22	52
300	2011-10-22	48
300	2012-10-22	18
300	2013-10-22	8
224	2014-10-22	54
224	2015-10-22	58

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

```sql
WITH Stages_crossed AS (
    SELECT P_ID, Start_time, Stages_crossed, SUM(Stages_crossed) OVER ( PARTITION BY P_ID ORDER BY Start_time ASC) AS Cumulative_Sum_of_Stages_Crossed,
    ROW_NUMBER() OVER ( PARTITION BY P_ID ORDER BY Start_time ASC) AS Row_Num
    FROM level_details
)
SELECT P_ID, Start_time, Stages_crossed, Cumulative_Sum_of_Stages_Crossed
FROM Stages_crossed
WHERE Row_Num > 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

P_ID	Start_time	Stages_crossed	Cumulative_Sum_of_Stages_Crossed
211	12-10-22 18:30	5	9
211	13-10-22 22:30	5	14
211	13-10-22 5:36	5	19
211	14-10-22 8:56	7	26
211	15-10-22 11:41	8	34
224	14-10-22 8:21	5	12
224	15-10-22 13:43	4	16
224	15-10-22 5:30	10	26
242	14-10-22 4:38	8	14
292	15-10-22 10:19	5	9
296	14-10-22 19:35	4	6
300	11-10-22 5:20	7	12
300	12-10-22 1:45	2	14
300	12-10-22 11:21	3	17
300	13-10-22 23:15	3	20
310	13-10-22 19:18	5	12
310	15-10-22 23:30	7	19
358	14-10-22 5:05	3	5
368	12-10-22 11:59	6	13
368	12-10-22 4:20	5	18
368	15-10-22 14:47	4	22
429	11-10-22 19:28	6	13

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

```sql
WITH Top_scores AS (
    SELECT P_ID, DEV_ID, SUM(Score) OVER ( PARTITION BY DEV_ID ORDER BY Score DESC) AS Total_Scores,
    RANK() OVER ( PARTITION BY DEV_ID ORDER BY Score) AS Ranking
    FROM level_details
)
SELECT P_ID, DEV_ID, Total_Scores
FROM Top_scores
WHERE Ranking <=3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

P_ID	DEV_ID	Total_Scores
663	bd_013	27110
632	bd_013	27110
300	bd_013	26910
428	bd_015	14630
483	bd_015	14250
224	bd_015	13200
211	bd_017	4540
644	bd_017	4150
590	bd_017	2400
663	rf_013	14930
632	rf_013	14930
641	rf_013	14730
641	rf_015	8510
644	rf_015	8470
292	rf_015	8320
656	rf_017	15160
211	rf_017	14880
429	rf_017	13780
663	wd_019	6675
558	wd_019	6575
590	wd_019	5940
358	zm_013	7180

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

```sql
SELECT P_ID, ROUND(AVG(Score),0) AS Average_Score, SUM(Score) AS Total_Score
FROM level_details
GROUP BY P_ID
HAVING Total_Score > Average_Score * 0.5
ORDER BY P_ID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

P_ID	Average_Score	Total_Score
211	1823	10940
224	4078	16310
242	3155	6310
292	1280	2560
296	570	1140
300	972	4860
310	4603	13810
319	50	50
358	95	190
368	2178	8710
428	380	380
429	3305	13220
483	3446	17230
547	1150	3450
558	635	635
590	1600	8000
632	2150	10750
641	127	380
644	750	2250
656	1205	4820
663	2150	10750
683	2591	18140

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.

```
call game_analysis.Top_n_headshots(5);
```

Result Grid | Filter Rows: | Export:

Dev_ID	Difficulty	Headshots_Count	ScoreRanking
bd_013	Medium	4	1
bd_013	Medium	8	2
bd_013	Medium	10	3
bd_013	Difficult	11	4
bd_013	Low	11	5

Call stored procedure game_analysis.Top_n_headshots

Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

n [IN] INT

Execute Cancel

```
DELIMITER //
CREATE PROCEDURE Top_n_headshots(IN n INT)
BEGIN
    SELECT Dev_ID, Difficulty, Headshots_Count,
    ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY Headshots_Count ASC) AS Score_Rank
    FROM level_details
    ORDER BY Dev_ID, Score_Rank LIMIT n;
END
DELIMITER ;
```

Conclusion

I Utilized MySQL Workbench, Microsoft Power Point, Canva, Microsoft Word, Notepad, GitHub and Draw.io to execute this project

I ensured that most of my queries were properly optimized, putting time and space complexity in check.

The entire 15 insights were carefully extracted for stakeholders to understand gaming behaviour

The insights generated could be used to improve the gaming industry and make future recommendations.

Used window functions like RANK() ROW_NUMBER(), SUM(), and OVER() to extract meaningful data.

Used stored procedure to execute complex SQL functions extracting insights.


Thank You!