DECODING GAMING BEHAVIOR

USING SQL

DHIVYA. V

INTRODUCTION:

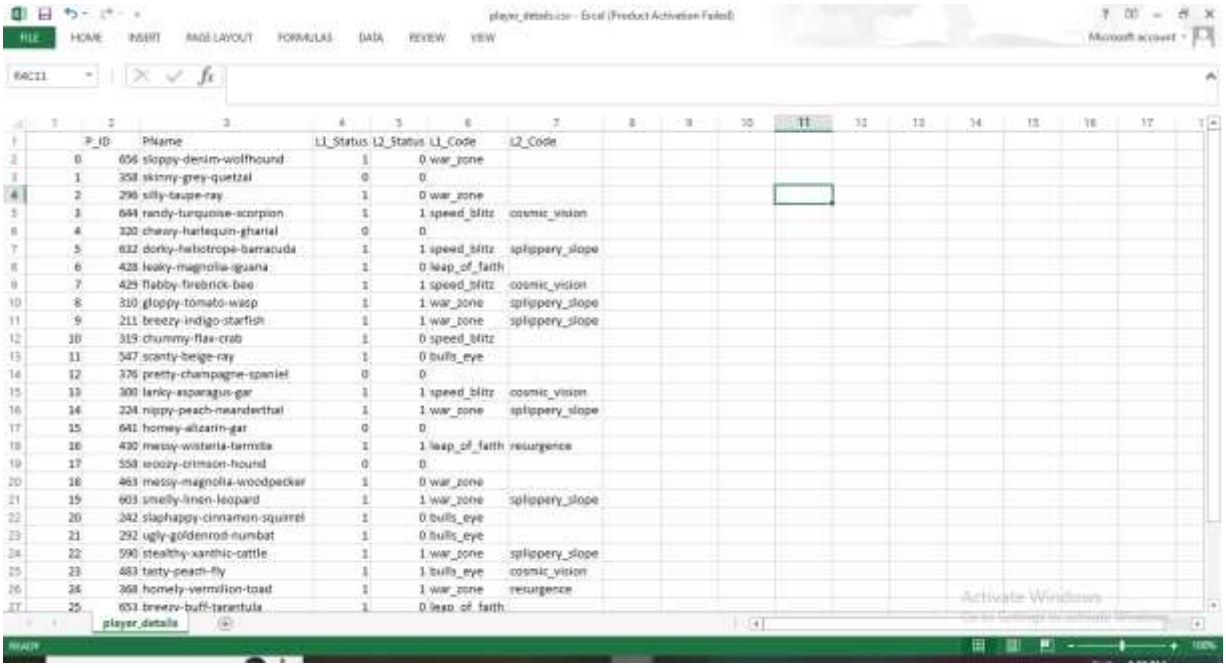
Decoding gaming behavior involves understanding the motivations, preferences, and patterns exhibited by individuals while engaging in gaming activities. Gaming behavior analysis encompasses various aspects, including player psychology, gameplay mechanics, social interactions, and game design elements. By decoding gaming behavior, researchers, game developers, and marketers seek to gain insights into player preferences, motivations, and decision-making processes within gaming environments.

TABLES:

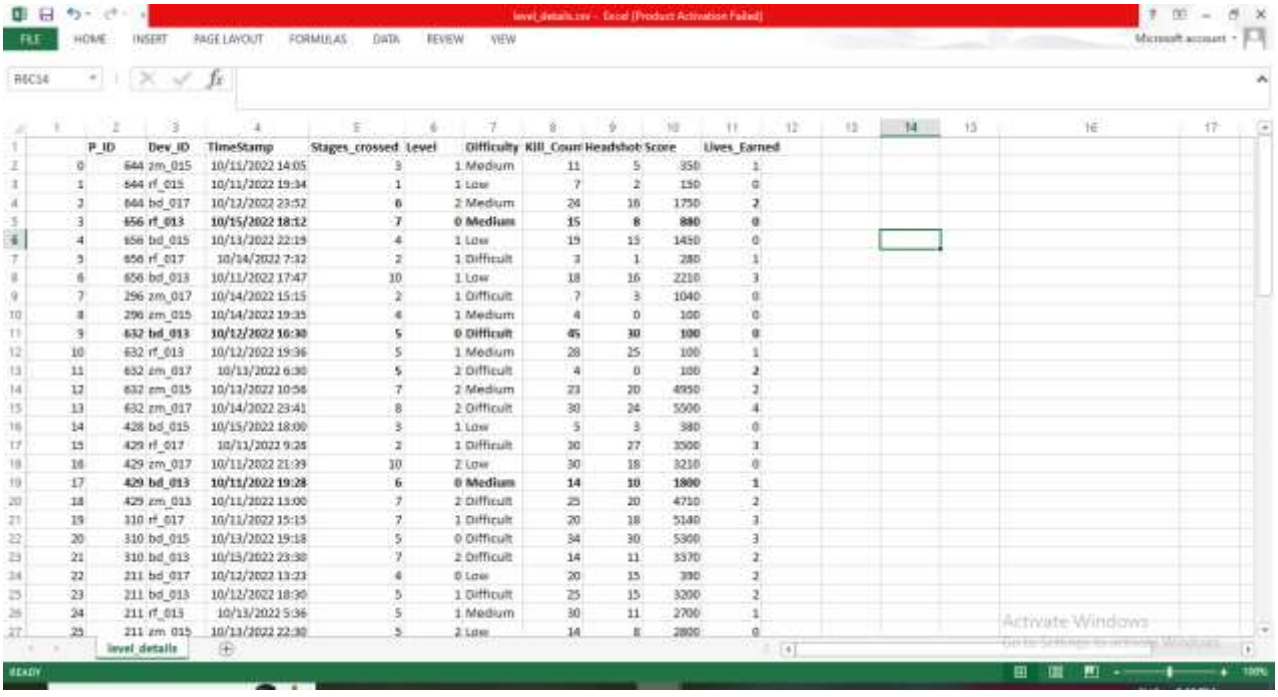
- Player Details Table:
  Description of each column (P_ID, PName, L1_status, L2_status, L1_code, L2_code).
- Level Details Table:
  Description of each column (P_ID, Dev_ID, start_time, stages_crossed, level, difficulty, kill_count, headshots_count, score, lives_earned).

DATA DESDRIPTION:

- Explanation of key terms and concepts in the dataset (e.g., level, difficulty, kill count).
- Overview of the relationships between the Player Details and Level Details tables.
- Discussion on how the data can be used to analyze gaming behavior.

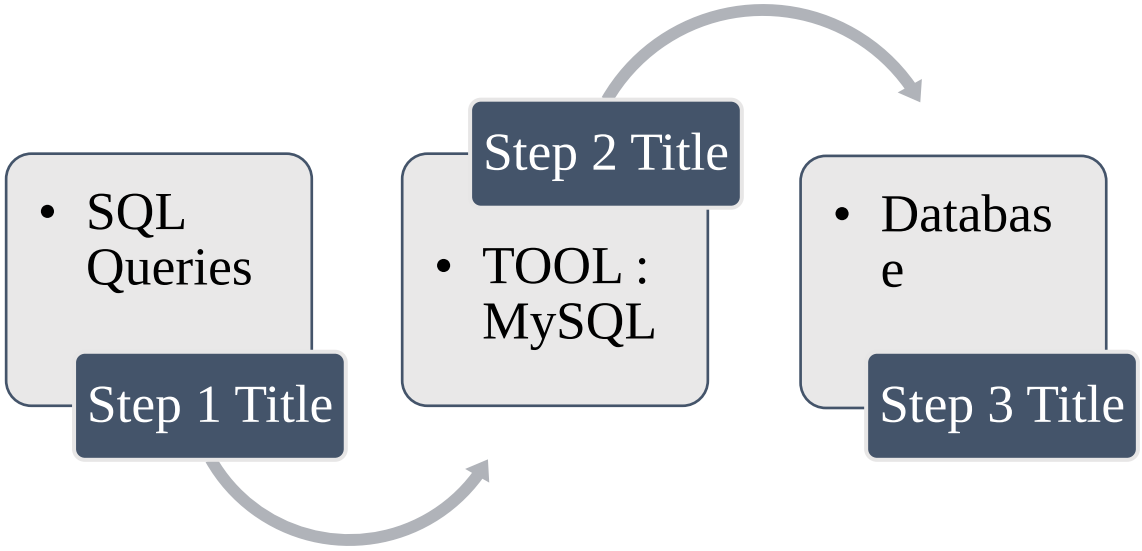


	P_ID	PName	L1_Status	L2_Status	L1_Code	L2_Code
0	656	sloppy-denim-wolfhound	1	0	war_zone	
1	358	skinny-grey-quetzal	0	0		
2	296	silly-taupe-ray	1	0	war_zone	
3	644	randy-turquoise-scorpion	1	1	speed_blitz	cosmic_vision
4	320	chewy-harlequin-gharial	0	0		
5	632	dorky-heliotrope-barracuda	1	1	speed_blitz	splippery_slope
6	428	leaky-magnolia-iguana	1	0	leap_of_faith	
7	429	flabby-firebrick-bee	1	1	speed_blitz	cosmic_vision
8	310	gloppy-tomato-wasp	1	1	war_zone	splippery_slope
9	211	breezy-indigo-starfish	1	1	war_zone	splippery_slope
10	319	chummy-flax-crab	1	0	speed_blitz	
11	547	scanty-beige-ray	1	0	bulls_eye	
12	376	pretty-champagne-spaniel	0	0		
13	300	lanky-asparagus-gar	1	1	speed_blitz	cosmic_vision
14	224	nippy-peach-neanderthal	1	1	war_zone	splippery_slope
15	641	homey-alizarin-gar	0	0		
16	430	messy-wisteria-termite	1	1	leap_of_faith	resurgence
17	558	woozy-crimson-hound	0	0		
18	463	messy-magnolia-woodpecker	1	0	war_zone	
19	603	smelly-linen-leopard	1	1	war_zone	splippery_slope
20	242	slaphappy-cinnamon-squirrel	1	0	bulls_eye	
21	292	ugly-goldenrod-numbat	1	0	bulls_eye	
22	590	stealthy-xanthic-cattle	1	1	war_zone	splippery_slope
23	483	tasty-peach-fly	1	1	bulls_eye	cosmic_vision
24	368	homely-vermilion-toad	1	1	war_zone	resurgence
25	653	breezy-buff-tarantula	1	0	leap_of_faith	



	P_ID	Dev_ID	TimeStamp	Stages_crossed	Level	Difficulty	Kill_Count	Headshots	Score	Lives_Earned
0	644	zm_015	10/11/2022 14:05	3	1	Medium	11	5	350	1
1	644	rf_015	10/11/2022 19:34	1	1	Low	7	2	150	0
2	644	bd_017	10/12/2022 23:52	6	2	Medium	24	16	1750	2
3	656	rf_013	10/15/2022 18:12	7	0	Medium	15	8	880	0
4	656	bd_015	10/13/2022 22:19	4	1	Low	19	15	1450	0
5	656	rf_017	10/14/2022 7:32	2	1	Difficult	3	1	280	1
6	656	bd_013	10/11/2022 17:47	10	1	Low	18	16	2210	3
7	296	zm_017	10/14/2022 15:15	2	1	Difficult	7	3	1040	0
8	296	zm_015	10/14/2022 19:35	4	1	Medium	4	0	100	0
9	632	bd_013	10/12/2022 16:30	5	0	Difficult	45	30	100	0
10	632	rf_013	10/12/2022 19:36	5	1	Medium	28	25	100	1
11	632	zm_017	10/13/2022 6:30	5	2	Difficult	4	0	100	2
12	632	zm_015	10/13/2022 10:56	7	2	Medium	23	20	4950	2
13	632	zm_017	10/14/2022 23:41	8	2	Difficult	30	24	5500	4
14	428	bd_015	10/15/2022 18:00	3	1	Low	5	3	380	0
15	429	rf_017	10/11/2022 9:28	2	1	Difficult	30	27	3500	3
16	429	zm_017	10/11/2022 21:39	10	2	Low	30	18	3210	0
17	429	bd_013	10/11/2022 19:28	6	0	Medium	14	10	1800	1
18	429	zm_013	10/11/2022 13:00	7	2	Difficult	25	20	4710	2
19	310	rf_017	10/11/2022 15:15	7	1	Difficult	20	18	5140	3
20	310	bd_015	10/13/2022 19:18	5	0	Difficult	34	30	5360	3
21	310	bd_013	10/13/2022 23:30	7	2	Difficult	14	11	3370	2
22	211	bd_017	10/12/2022 13:23	4	0	Low	20	15	390	2
23	211	bd_013	10/12/2022 18:30	5	1	Difficult	25	15	3200	2
24	211	rf_013	10/13/2022 5:36	5	1	Medium	30	11	2700	1
25	211	zm_015	10/13/2022 22:30	5	2	Low	14	8	2800	0

STEPS INVOLVED:



TOOLS USED: MYSQL

- MySQL's relational database model is well-suited for storing structured gaming data, such as player details and game levels, in a systematic and organized manner.
- SQL queries can be used to extract, analyze, and manipulate gaming data stored in MySQL databases, facilitating data analysis and insights generation.
- MySQL's scalability allows for the storage and management of large volumes of gaming data, accommodating the requirements of diverse gaming projects and applications.
- Security features provided by MySQL ensure the confidentiality, integrity, and availability of gaming data, safeguarding it from unauthorized access and data breaches.

CREATE TABLE:

```
CREATE DATABASE GAME_ANALYSIS

USE GAME_ANALYSIS;
```

ALTER TABLE:

```
alter table  player_details  modify L1_Status varchar(30);

alter table player_details  modify L2_Status varchar(30);

alter table player_details modify P_ID int primary key;

alter table player_details drop myunknowncolumn;
```

alter table level_details drop myunknowncolumn;

alter table level_details change timestamp start_datetime datetime;

alter table level_details modify Dev_Id varchar(10);

alter table level_details modify Difficulty varchar(15);

alter table level_details add primary key(P_ID,Dev_id,start_datetime);

DISPLAY TABLE:

SELECT * FROM player_details;

SELECT * FROM level_details;

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

```
SELECT ld.P_ID, ld.Dev_ID, pd.PName, ld.Difficulty AS Difficulty_level
FROM level_details ld
JOIN player_details pd ON ld.P_ID = pd.P_ID
WHERE ld.Level = 0;
```

P_ID	Dev_ID	PName	Difficulty_level
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

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

stages are crossed.

SELECT pd.L1_Code, AVG(ld.Kill_Count) AS Avg_Kill_Count

FROM level_details ld

JOIN player_details pd ON ld.P_ID = pd.P_ID

WHERE ld.Lives_Earned = 2 AND ld.Stages_crossed >= 3

GROUP BY pd.L1_Code;

	L1_Code	Avg_Kill_Count
▶	war_zone	19.2857
	bulls_eye	22.2500
	speed_blitz	19.3333

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_details ld
JOIN
   player_details pd ON ld.P_ID = pd.P_ID
WHERE
   ld.Level = 2
   AND ld.Dev_ID LIKE 'zm_%'
GROUP BY
   ld.Difficulty
ORDER BY
   Total_Stages_Crossed DESC;
```

	Difficulty	Total_Stages_Crossed
▶	Difficult	46
	Medium	35
	Low	15

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

```
SELECT ld.P_ID,
    COUNT(DISTINCT DATE(ld.start_datetime)) AS Total_Unique_Dates
FROM level_details ld
GROUP BY ld.P_ID
HAVING COUNT(DISTINCT DATE(ld.start_datetime)) > 1;
```

P_ID	Total_Unique_Dates
211	4
224	2
242	2
292	2
300	3
310	3
368	2
483	3
590	3
632	3
641	2
644	2

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

```
SELECT
  ld.P_ID,
  ld.Level,
  SUM(ld.Kill_Count) AS Levelwise_Sum_of_Kill_Counts
FROM
  level_details ld
WHERE
  ld.Kill_Count > (
    SELECT
      AVG(ld2.Kill_Count)
```

```
    FROM
        level_details ld2
    WHERE
        ld2.Difficulty = 'Medium'
  )
GROUP BY
  ld.P_ID, ld.Level;
```

P_ID	Level	Levelwise_Sum_of_Kill_Counts
211	1	55
211	0	20
224	2	58
224	1	54
242	1	58
292	1	21
300	1	48
310	0	34
310	1	20
368	2	24
368	1	20
429	1	30

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

```
SELECT
  ld.Level,
  pd.L1_Code AS Level_Code,
  SUM(ld.Lives_Earned) AS Total_Lives_Earned
FROM
  level_details ld
JOIN
  player_details pd ON ld.P_ID = pd.P_ID
WHERE
  ld.Level > 0
```

```
GROUP BY
    ld.Level, pd.L1_Code
ORDER BY
    ld.Level ASC;
```

Level	Level_Code	Total_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
2	war_zone	17

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
        ld.P_ID,
        ld.Dev_ID,
        ld.Score,
        ld.Difficulty,
        ROW_NUMBER() OVER (PARTITION BY ld.Dev_ID ORDER BY ld.Score ASC) AS ScoreRank
    FROM
        level_details ld
)
SELECT
    Dev_ID,
    Score,
    Difficulty
FROM
```

```
    RankedScores
WHERE
    ScoreRank <= 3;
```

Dev_ID	Score	Difficulty
bd_013	100	Difficult
bd_013	100	Difficult
bd_013	540	Low
bd_015	380	Low
bd_015	1050	Medium
bd_015	1300	Difficult
bd_017	390	Low
bd_017	1750	Medium
bd_017	2400	Low
rf_013	100	Medium
rf_013	100	Medium
rf_013	105	Low

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

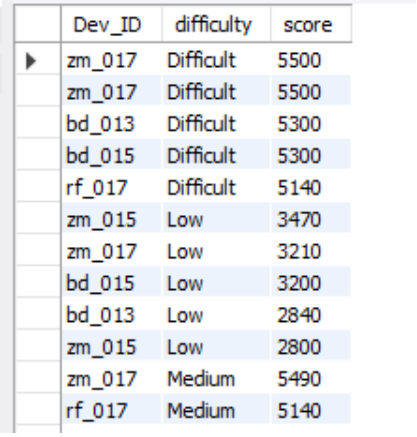
```
SELECT Dev_ID, MIN(start_datetime) AS first_login
FROM level_details
GROUP BY Dev_ID;
```

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
zm_013	2022-10-11 13:00:22
wd_019	2022-10-12 23:19:17

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 ld.Dev_ID,
        ld.difficulty,
```

```
        ld.score,
        RANK() OVER (PARTITION BY ld.difficulty ORDER BY ld.score DESC) AS score_rank
    FROM level_details ld
)
SELECT Dev_ID,
    difficulty,
    score
FROM RankedScores
WHERE score_rank <= 5;
```



Dev_ID	difficulty	score
zm_017	Difficult	5500
zm_017	Difficult	5500
bd_013	Difficult	5300
bd_015	Difficult	5300
rf_017	Difficult	5140
zm_015	Low	3470
zm_017	Low	3210
bd_015	Low	3200
bd_013	Low	2840
zm_015	Low	2800
zm_017	Medium	5490
rf_017	Medium	5140

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 ld.P_ID,
    ld.Dev_ID,
    ld.start_datetime AS first_login_datetime
FROM level_details ld
JOIN (
    SELECT P_ID,
        MIN(start_datetime) AS first_login_time
```

```
   FROM level_details
   GROUP BY P_ID
) AS first_login ON ld.P_ID = first_login.P_ID AND ld.start_datetime = first_login.first_login_time;
```

P_ID	Dev_ID	first_login_datetime
211	bd_017	2022-10-12 13:23:45
224	rf_017	2022-10-14 01:15:56
242	bd_013	2022-10-13 01:14:29
292	rf_013	2022-10-12 04:29:45
296	zm_017	2022-10-14 15:15:15
300	rf_013	2022-10-11 05:20:40
310	rf_017	2022-10-11 15:15:15
319	zm_017	2022-10-12 14:20:40
358	zm_017	2022-10-14 05:05:05
368	zm_015	2022-10-12 01:14:34
428	bd_015	2022-10-15 18:00:00
429	rf_017	2022-10-11 09:28:56

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

a) **Using window functions**

```
SELECT P_ID,
    start_datetime,
    kill_count,
    SUM(kill_count) OVER (PARTITION BY P_ID ORDER BY start_datetime) AS cumulative_kill_count
FROM level_details;
```

P_ID	start_datetime	kill_count	cumulative_kill_count
211	2022-10-12 13:23:45	20	20
211	2022-10-12 18:30:30	25	45
211	2022-10-13 05:36:15	30	75
211	2022-10-13 22:30:18	14	89
211	2022-10-14 08:56:24	9	98
211	2022-10-15 11:41:19	15	113
224	2022-10-14 01:15:56	20	20
224	2022-10-14 08:21:49	34	54
224	2022-10-15 05:30:28	30	84
224	2022-10-15 13:43:50	28	112
242	2022-10-13 01:14:29	21	21
242	2022-10-14 04:38:50	37	58

b) Without window functions

```
SELECT ld.P_ID,
    ld.start_datetime,
    ld.kill_count,
    (SELECT SUM(kill_count)
     FROM level_details ld2
     WHERE ld2.P_ID = ld.P_ID AND ld2.start_datetime <= ld.start_datetime) AS cumulative_kill_count
FROM level_details ld;
```

P_ID	start_datetime	kill_count	cumulative_kill_count
211	2022-10-12 18:30:30	25	45
211	2022-10-12 13:23:45	20	20
211	2022-10-13 05:36:15	30	75
211	2022-10-15 11:41:19	15	113
211	2022-10-13 22:30:18	14	89
211	2022-10-14 08:56:24	9	98
224	2022-10-15 05:30:28	30	84
224	2022-10-15 13:43:50	28	112
224	2022-10-14 08:21:49	34	54
224	2022-10-14 01:15:56	20	20
242	2022-10-13 01:14:29	21	21
242	2022-10-14 04:38:50	37	58

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

excluding the most recent `start_datetime`.

```
WITH CumulativeSum AS (
   SELECT P_ID,
       start_datetime,
       stages_crossed,
       SUM(stages_crossed) OVER (PARTITION BY P_ID ORDER BY start_datetime) AS cumulative_stages
   FROM level_details
)
SELECT P_ID,
     start_datetime,
     stages_crossed,
```

cumulative_stages - LAG(stages_crossed, 1, 0) OVER (PARTITION BY P_ID ORDER BY start_datetime) AS cumulative_sum_excluding_latest

FROM CumulativeSum;

P_ID	start_datetime	stages_crossed	cumulative_sum_excluding_late
211	2022-10-12 13:23:45	4	4
211	2022-10-12 18:30:30	5	5
211	2022-10-13 05:36:15	5	9
211	2022-10-13 22:30:18	5	14
211	2022-10-14 08:56:24	7	21
211	2022-10-15 11:41:19	8	27
224	2022-10-14 01:15:56	7	7
224	2022-10-14 08:21:49	5	5
224	2022-10-15 05:30:28	10	17
224	2022-10-15 13:43:50	4	16
242	2022-10-13 01:14:29	6	6
242	2022-10-14 04:38:50	8	8

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

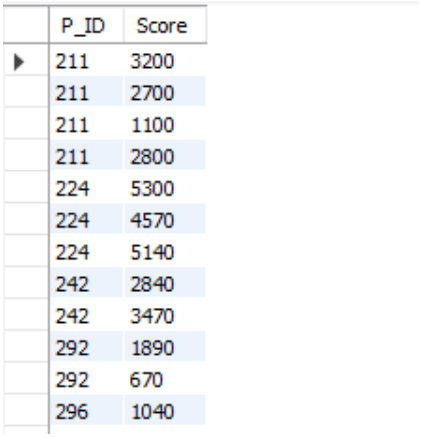
```
WITH RankedScores AS (
   SELECT Dev_ID,
        P_ID,
        SUM(score) AS total_score,
        RANK() OVER (PARTITION BY Dev_ID ORDER BY SUM(score) DESC) AS score_rank
   FROM level_details
   GROUP BY Dev_ID, P_ID
)
SELECT Dev_ID,
     P_ID,
     total_score
FROM RankedScores
WHERE score_rank <= 3;
```

Dev_ID	P_ID	total_score
bd_013	224	9870
bd_013	310	3370
bd_013	211	3200
bd_015	310	5300
bd_015	683	3200
bd_015	368	1950
bd_017	590	2400
bd_017	644	1750
bd_017	211	390
rf_013	368	2970
rf_013	211	2700
rf_013	300	2300

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

```
WITH PlayerAverage AS (
    SELECT
        P_ID,
        AVG(Score) AS AverageScore
    FROM
        level_details
    GROUP BY
        P_ID
)
SELECT
    ld.P_ID,
    ld.Score
FROM
    level_details ld
JOIN
    PlayerAverage pa ON ld.P_ID = pa.P_ID
WHERE
```

ld.Score > 0.5 * pa.AverageScore;



P_ID	Score
211	3200
211	2700
211	1100
211	2800
224	5300
224	4570
224	5140
242	2840
242	3470
292	1890
292	670
296	1040

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 GetTopNHeadshots(IN n INT)
BEGIN
   SET @n = n;


   SELECT Dev_ID,
        difficulty,
        headshots_count,
        headshots_rank
   FROM (
      SELECT Dev_ID,
           difficulty,
           headshots_count,
           ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY headshots_count ASC) AS headshots_rank
      FROM level_details
```

```
    ) AS ranked_headshots
    WHERE headshots_rank <= @n;
END //

DELIMITER ;

CALL GetTopNHeadshots(5);
```

Dev_ID	difficulty	headshots_count	headshots_rank
bd_013	Medium	4	1
bd_013	Medium	8	2
bd_013	Medium	10	3
bd_013	Low	11	4
bd_013	Difficult	11	5
bd_015	Low	3	1

FUTURE OUTCOME

1. Precision Personalization: Tailoring gaming experiences to individual preferences and emotional responses through advanced AI and player analytics.
2. Enhanced Immersion: Utilizing VR, AR, and immersive technologies to engage players on deeper emotional, cognitive, and physiological levels.
3. Ethical Design and Regulation: Developing ethical design principles and regulatory frameworks to promote player well-being and mitigate potential harms.
4. Cross-Disciplinary Insights: Fostering collaboration across psychology, neuroscience, sociology, and anthropology for deeper understandings of gaming behavior.
5. Positive Social Impact: Harnessing gaming for education, healthcare, and social change, fostering empathy, understanding, and societal progress.

CONCLUSION

- In conclusion, understanding gaming behavior is crucial for creating better gaming experiences. By analyzing player data, we've learned about player preferences and patterns.
- The dataset has given us insights into which game levels and difficulties players prefer. This information can help improve game design to better meet player needs.
- Looking ahead, this project's findings will lead to more engaging games and smarter marketing strategies. By knowing what players want, we can create games that players love and market them more effectively.