GAME ANALYSIS:

Decoding Gaming Behavior

Presentation by Matthew Adenyo

Mentorness Internship | 2024 | Batch Name: MIP-DA-06

INTRODUCTION

This presentation covers task two of the Mentorness Internship Program.

It focused on working with a dataset related to a game. The dataset includes two tables: “Player Details” and “Level Details”.

The aim is to answer 17 questions by writing SQL queries.

PROBLEM STATEMENT

- Players play a game divided into 3-levels (L0, L1 and L2)
- Each level has 3 difficulty levels (Low, Medium, High)
- At each level, players have to kill the opponents using guns/physical fight
- Each level has multiple stages at each difficulty level.
- A player can only play L1 using its system-generated L1_code.
- Only players who have played Level 1 can possibly play Level 2 using its system-generated L2_code.

- **By default, a player can play L0.**
- **Each player can login to the game using a Dev_ID.**
- **Players can earn extra lives at each stage in a level.**

DATASET DESCRIPTION

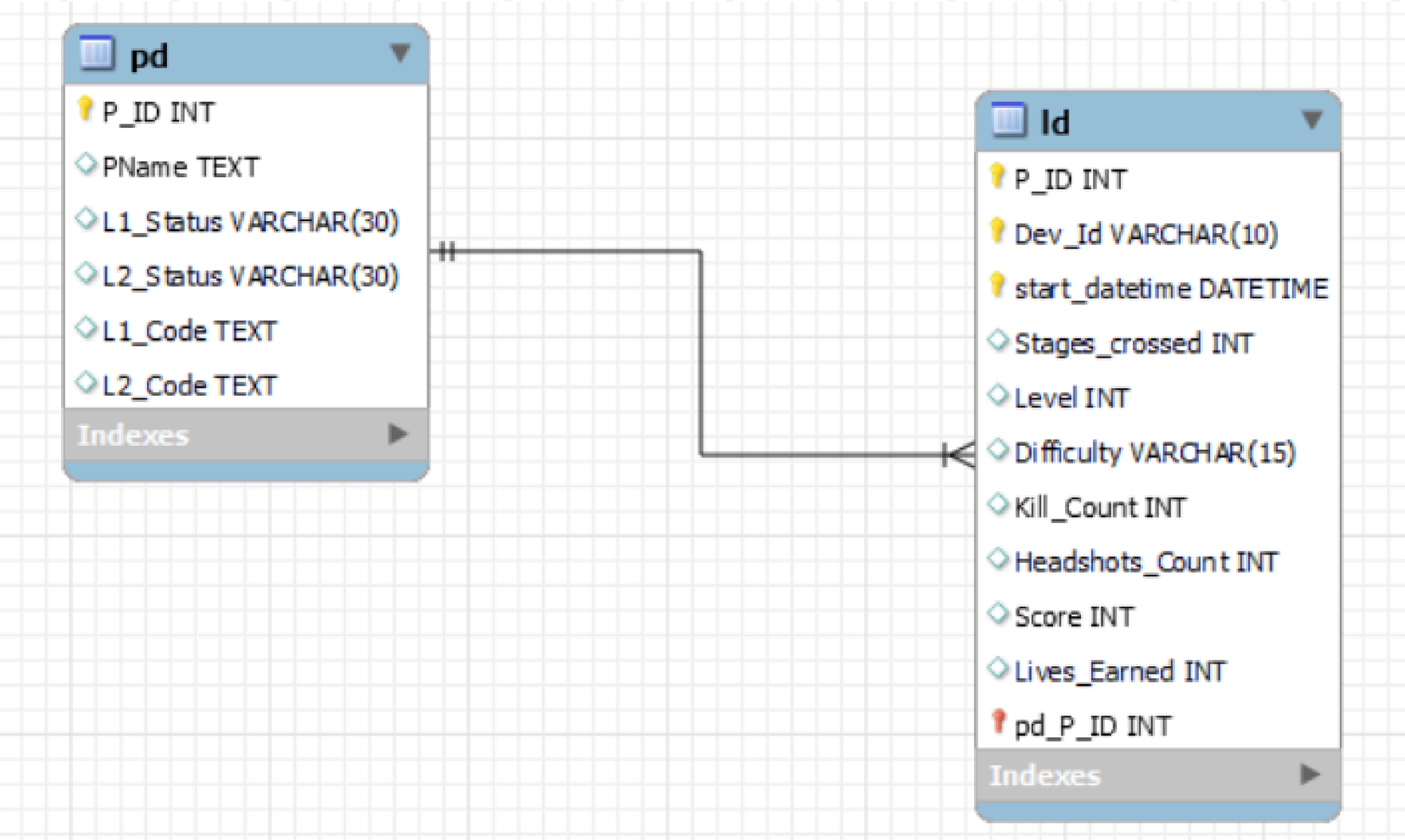
PLAYER DETAILS TABLE

- 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



Presentation by Matthew Adenyo | Mentorness Internship | 2024

DATA CLEANING & TRANSFORMATION

To kick-start the analysis, I changed the table names, modified and dropped irrelevant columns.

```sql
RENAME TABLE player_details TO pd;
RENAME TABLE level_details2 TO ld;
```

```sql
alter table pd modify L1_Status varchar(30);
alter table pd modify L2_Status varchar(30);
alter table pd modify P_ID int primary key;
alter table pd drop myunknowncolumn;
```

```sql
alter table ld drop myunknowncolumn;
alter table ld change timestamp start_datetime datetime;
alter table ld modify Dev_Id varchar(10);
alter table ld modify Difficulty varchar(15);
alter table ld add primary key(P_ID,Dev_id,start_datetime);
```

ANALYSIS: QUERIES & RESULTS

Q1) Extract P_ID, Dev_ID, PName and Difficulty_level of all players at level 0

```sql
SELECT pd.P_ID, ld.Dev_ID, pd.PName, ld.Difficulty
FROM pd
JOIN ld
ON pd.P_ID = ld.P_ID
WHERE ld.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
632	bd_013	dorky-heliotrope-barracuda	Difficult

Q2) Find Level1_code wise Avg_Kill_Count where lives_earned is 2 and at least 3 stages are crossed

```sql
SELECT pd.L1_Code, AVG(ld.Kill_Count) AS Avg_Kill_Count
FROM pd
JOIN ld
ON pd.P_ID = ld.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

Q3) Find the total number of stages crossed at each difficulty level where for Level2 with players use zm_series devices. Arrange the result in decreasing order of total number of stages crossed.

```sql
SELECT Difficulty AS Difficulty_Level, COUNT(Stages_Crossed) AS Total_Stages_Crossed
FROM ld
WHERE Level = 2 AND Dev_Id LIKE 'zm%'
GROUP BY Difficulty_Level
ORDER BY Total_Stages_Crossed DESC;
```

Difficulty_Level	Total_Stages_Crossed
Difficult	7
Medium	6
Low	2

Q4) Extract P_ID and the total number of unique dates for those players who have played games on multiple days.

```sql
SELECT P_ID, COUNT(DISTINCT(Start_datetime)) AS Unique_Dates
FROM ld
GROUP BY P_ID
HAVING COUNT(DISTINCT(Start_datetime)) > 1
ORDER BY Unique_Dates DESC;
```

P_ID	Unique_Dates
683	7
211	6
300	5
483	5
590	5
632	5
663	5
224	4
368	4

Q5) Find P_ID and level-wise sum of kill_counts where kill_count is greater than avg kill count for the Medium difficulty.

```sql
SELECT P_ID, Level, SUM(Kill_Count) AS Total_Kill_Counts, AVG(Kill_Count) AS Avg_Kill_Count
FROM ld
WHERE Difficulty = 'Medium'
GROUP BY P_ID, Level
HAVING Total_Kill_Counts > Avg_Kill_Count
ORDER BY Level DESC;
```

P_ID	Level	Total_Kill_Counts	Avg_Kill_Count
300	2	12	6.0000
590	2	24	12.0000
483	1	40	20.0000

Q6) Find Level and its corresponding Level code-wise sum of lives earned excluding level 0. Arrange in ascending order of level.

```sql
SELECT ld.Level, SUM(ld.Lives_Earned) OVER (ORDER BY ld.Level) AS Total_Lives, pd.L1_Code, pd.L2_Code
FROM ld
JOIN pd
ON ld.P_ID = pd.P_ID
WHERE ld.Level <> 0;
```

Level	Total_Lives	L1_Code	L2_Code
1	23	war_zone	splippery_slope
1	23	war_zone	splippery_slope
1	23	leap_of_faith	
1	23	war_zone	splippery_slope
1	23	speed_blitz	
1	23	speed_blitz	cosmic_vision
1	23	bulls_eye	cosmic_vision
1	23	war_zone	splippery_slope
1	23	war_zone	splippery_slope

Q7) Find Top 3 score based on each dev_id and Rank them in increasing order using Row_Number. Display difficulty as well.

```sql
WITH Ranked_Scores AS (
    SELECT Dev_Id, Score, Difficulty,
        ROW_NUMBER() OVER (PARTITION BY Dev_Id ORDER BY Score DESC) AS Ranks
    FROM ld
)
SELECT Dev_Id, Score, Difficulty, Ranks
FROM Ranked_Scores
WHERE Ranks <= 3
ORDER BY Dev_Id, Ranks;
```

Dev_Id	Score	Difficulty	Ranks
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

Q8) Find first_login datetime for each device id

```sql
SELECT Dev_ID, MIN(Start_datetime) AS First_Login_Datetime
FROM ld
GROUP BY Dev_ID;
```

Dev_ID	First_Login_Datetime
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

Q9) Find Top 5 scores based on each difficulty level and Rank them in increasing order using Rank. Display dev_id as well.

```sql
WITH Ranked_Scores AS (
    SELECT Dev_Id, Score, Difficulty,
        RANK() OVER (PARTITION BY Difficulty ORDER BY Score DESC) AS Ranks
    FROM ld
)
SELECT Dev_Id, Score, Difficulty, Ranks
FROM Ranked_Scores
WHERE Ranks <= 5
ORDER BY Difficulty, Ranks;
```

Dev_Id	Score	Difficulty	Ranks
zm_017	5500	Difficult	1
zm_017	5500	Difficult	1
bd_013	5300	Difficult	3
bd_015	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

3

Q10) 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.

```sql
SELECT ld.P_Id, ld.Dev_Id, ld.Start_datetime AS First_Login_Datetime
FROM ld
INNER JOIN (
    SELECT P_Id, MIN(Start_datetime) AS Min_Start_Datetime
    FROM ld
    GROUP BY p_id
) AS Min_Start_Times
ON ld.P_Id = Min_Start_Times.P_Id AND ld.Start_datetime = Min_Start_Times.Min_Start_Datetime;
```

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

Q11) 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

b) without window function

```sql
-- a) window function
SELECT ld.P_ID, pd.PName, ld.Start_datetime,
    SUM(ld.Kill_Count) OVER (PARTITION BY ld.P_ID ORDER BY ld.Start_datetime) AS Games_Played_So_Far
FROM ld
JOIN pd
ON ld.P_ID = pd.P_ID;
```

P_ID	PName	Start_datetime	Games_Played_So_Far
211	breezy-indigo-starfish	2022-10-12 13:23:45	20
211	breezy-indigo-starfish	2022-10-12 18:30:30	45
211	breezy-indigo-starfish	2022-10-13 05:36:15	75
211	breezy-indigo-starfish	2022-10-13 22:30:18	89
211	breezy-indigo-starfish	2022-10-14 08:56:24	98
211	breezy-indigo-starfish	2022-10-15 11:41:19	113

```sql
-- b) without window function
SELECT ld.P_ID, pd.PName, ld.Start_datetime, SUM(ld2.Kill_Count) AS Games_Played_So_Far
FROM ld
JOIN pd
ON ld.P_ID = pd.P_ID
JOIN ld AS ld2
ON ld.P_ID = ld2.P_ID AND ld2.Start_datetime <= ld.Start_datetime
GROUP BY ld.P_ID, ld.Start_datetime
ORDER BY ld.P_ID, ld.Start_datetime;
```

P_ID	PName	Start_datetime	Games_Played_So_Far
211	breezy-indigo-starfish	2022-10-12 13:23:45	20
211	breezy-indigo-starfish	2022-10-12 18:30:30	45
211	breezy-indigo-starfish	2022-10-13 05:36:15	75
211	breezy-indigo-starfish	2022-10-13 22:30:18	89
211	breezy-indigo-starfish	2022-10-14 08:56:24	98
211	breezy-indigo-starfish	2022-10-15 11:41:19	113
224	nippy-peach-neanderthal	2022-10-14 01:15:56	20

Q12) Find the cumulative sum of stages crossed over a start_datetime

```sql
SELECT Start_datetime, SUM(Stages_crossed) OVER (ORDER BY Start_datetime) AS Cumulative_Sum_of_Stage_Crossed
FROM ld;
```

Start_datetime	Cumulative_Sum_of_Stage_Crossed
2022-10-11 02:23:45	4
2022-10-11 05:20:40	11
2022-10-11 09:28:56	13
2022-10-11 13:00:22	20
2022-10-11 14:05:08	23
2022-10-11 14:33:27	33
2022-10-11 15:15:15	40
2022-10-11 17:47:09	50
2022-10-11 18:45:55	53
2022-10-11 19:19:19	58

Q13) Find the cumulative sum of an stages crossed over a start_datetime for each player id but exclude the most recent start_datetime

```sql
WITH Ranked_Stages AS (
    SELECT P_ID, Start_datetime, Stages_crossed,
        ROW_NUMBER() OVER (PARTITION BY P_ID ORDER BY Start_datetime DESC) AS rn
    FROM ld
)
SELECT P_ID, Start_datetime, Stages_crossed,
    SUM(Stages_crossed) OVER (PARTITION BY P_ID ORDER BY Start_datetime) - Stages_crossed AS cumulative_sum
FROM Ranked_Stages
WHERE rn > 1;
```

P_ID	Start_datetime	Stages_crossed	cumulative_sum
211	2022-10-12 13:23:45	4	0
211	2022-10-12 18:30:30	5	4
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	19
224	2022-10-14 01:15:56	7	0
224	2022-10-14 08:21:49	5	7
224	2022-10-15 05:30:28	10	12

Q14) Extract top 3 highest sum of score for each device id and the corresponding player_id

```
SELECT P_ID, Dev_Id, SUM(Score) AS Sum_of_Scores
FROM ld
GROUP BY P_ID, Dev_ID
ORDER BY Sum_of_Scores DESC
LIMIT 3;
```

P_ID	Dev_Id	Sum_of_Scores
224	bd_013	9870
683	zm_017	8900
632	zm_017	5600

Q15) Find players who scored more than 50% of the avg score scored by sum of scores for each player_id

```sql
SELECT ld.P_ID, pd.PName, SUM(ld.Score) AS Total_Scores, ROUND(AVG(ld.Score), 0) AS Avg_Scores
FROM ld
JOIN pd
ON ld.P_ID = pd.P_ID
GROUP BY P_ID
HAVING SUM(ld.Score) > 0.5 * ROUND(AVG(ld.Score), 0);
```

P_ID	PName	Total_Scores	Avg_Scores
211	breezy-indigo-starfish	10940	1823
224	nippy-peach-neanderthal	16310	4078
242	slaphappy-cinnamon-squirrel	6310	3155
292	ugly-goldenrod-numbat	2560	1280
296	silly-taupe-ray	1140	570
300	lanky-asparagus-gar	4860	972
310	gloppy-tomato-wasp	13810	4603
319	chummy-flax-crab	50	50

Q16) Create a stored procedure to find top n headshots_count based on each dev_id and Rank them in increasing order. using Row_Number. Display difficulty as well.

```
DELIMITER //
CREATE PROCEDURE Get_Top_N_Headshots(IN n INT)
BEGIN
    SELECT Dev_ID, Headshots_Count, Difficulty,
           ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY Headshots_Count DESC) AS `Rank`
    FROM ld
    ORDER BY Dev_ID, `Rank`
    LIMIT n;
END //
DELIMITER ;

CALL Get_Top_N_Headshots(15);
```

Dev_ID	Headshots_Count	Difficulty	Rank
bd_013	30	Difficult	1
bd_013	30	Difficult	2
bd_013	25	Difficult	3
bd_013	22	Difficult	4
bd_013	17	Low	5
bd_013	16	Low	6
bd_013	15	Difficult	7
bd_013	11	Low	8
bd_013	11	Difficult	9
bd_013	11	Low	10
bd_013	10	Medium	11

Q17) Create a function to return sum of Score for a given player_id.

```
DELIMITER //
CREATE FUNCTION Get_Total_Score_For_Player(player_id INT) RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE total_score INT;

    SELECT SUM(Score) INTO total_score
    FROM ld
    WHERE P_ID = player_id;

    RETURN total_score;
END//
DELIMITER ;

SELECT Get_Total_Score_For_Player(211) as Total_Score;
```

Total_Score
10940

CONCLUSION

The analysis of the game data using SQL has provided valuable insights into various aspects of the game's performance and player behaviour. By querying and manipulating the data, I was able to uncover trends and patterns that shed light on key aspects of the game.

KEY AREAS COVERED:

- Sub-Queries
- Aggregate Functions
- Window Functions
- Store Procedure

THANK YOU

Presentation by Matthew Adenyo

You are appreciated

Mentorness Internship | 2024