GAME ANALYSIS WITH SQL

MENTORNESS
INTENSHIP PROJECT
BY
JAINIL GOHEL

CONTENTS

PROJECT OVERVIEW

IN THIS PROJECT, I WILL BE WORKING WITH A DATASET RELATED TO A GAME. THE DATASET INCLUDES TWO TABLES: `PLAYER DETAILS` AND `LEVEL DETAILS`. THERE ARE 15 QUESTIONS FOR WHICH I HAVE FOUND THE ANSWERS BY WRITING SQL QUERIES.

DATA DESCRIPTION

PLAYER DETAILS TABLE:

- `P_ID`: PLAYER ID
- `PNAME`: PLAYER NAME
- `L1_STATUS`: LEVEL 1 STATUS
- `L2_STATUS`: LEVEL 2 STATUS
- `L1_CODE`: SYSTEMGENERATED LEVEL 1 CODE
- `L2_CODE`: SYSTEMGENERATED LEVEL 2 COD

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 EARNEDT

QUERIES AND OUTPUT

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

```
select l.P_ID,l.MyUnknownColumn,p.PName,l.Difficult
from player_details p
join level_details2 l
on p.P_ID=l.P_ID where l.level = 0
order by p.P_ID desc
```

P_ID	MyUnknownColumn	PName	Difficulty
656	3	sloppy-denim-wolfhound	Medium
641	73	homey-alizarin-gar	Low
641	74	homey-alizarin-gar	Medium
641	75	homey-alizarin-gar	Difficult
632	9	dorky-heliotrope-barracuda	Difficult
558	76	woozy-crimson-hound	Difficult
429	17	flabby-firebrick-bee	Medium
358	71	skinny-grey-quetzal	Low
358	72	skinny-grey-quetzal	Medium
310	20	gloppy-tomato-wasp	Difficult
300	34	lanky-asparagus-gar	Difficult
211	22	breezy-indigo-starfish	Low

QUERIES AND OUTPUT

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

```
select p.L1_Code,AVG(l.Kill_Count) AS Avg_Kills
from player_details p
join level_details2 l
on p.P_ID=l.P_ID
where l.Lives_Earned=2 and l.Stages_crossed >=3
group by p.L1_Code
```

L1_Code	Avg_Kills
bulls_eye	22.2500
speed_blitz	19.3333
war_zone	19.2857

QUERIES AND OUTPUT

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

```
SELECT Difficulty AS Diff_Level, COUNT(Stages_crossed) AS Total_Stages
FROM level_details2
WHERE Level = 2 AND Dev_ID LIKE 'zm%'
GROUP BY Difficulty
ORDER BY Total_Stages DESC
```

Diff_Level	Total_Stages
Difficult	7
Medium	6
Low	2

QUERIES AND OUTPUT

4. EXTRACT `P_ID` AND THE TOTAL NUMBER OF UNIQUE DATES FOR THOSE PLAYERS WHO HAVE PLAYEDGAMES ON MULTIPLE DAYS.

```
select P_ID,count(distinct(Start_datetime)) as Unique_Dates
from level_details2
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
590	5
632	5
483	5
663	5
656	4
368	4
429	4
224	4
310	3
547	3

QUERIES AND OUTPUT

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,Level,Sum(kill_Count) as Total,AVG(kill_Count) AS Avg_kills
from level_details2
where Difficulty='Medium'
group by P_ID,Level
having Sum(kill_Count)>AVG(kill_Count)
order by Level desc
```

P_ID	Level	Total	Avg_kills
300	2	12	6.0000
590	2	24	12.0000
483	1	40	20.0000

QUERIES AND OUTPUT

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, L1_Code,
(SELECT SUM(Lives_Earned) FROM level_details2 WHERE Level <= l.Level) AS totallives
FROM player_details p
JOIN level_details2 l ON p.P_ID = l.P_ID
WHERE l.Level <> 0
```

Level	L1_Code	L1_Code	totallives
1	speed_blitz	speed_blitz	29
1	speed_blitz	speed_blitz	29
2	speed_blitz	speed_blitz	80
1	war_zone	war_zone	29
1	war_zone	war_zone	29
1	war_zone	war_zone	29
1	war_zone	war_zone	29
1	war_zone	war_zone	29
1	war_zone	war_zone	29
1	speed_blitz	speed_blitz	29
2	speed_blitz	speed_blitz	80
2	speed_blitz	speed_blitz	80
2	speed_blitz	speed_blitz	80
1	leap of f...	leap of f...	29

QUERIES AND OUTPUT

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.

```
select Score,MyUnknownColumn,ROW_NUMBER() over (partition by MyUnknownColumn order by Score) as rownum
from level_details2
select Score,MyUnknownColumn,RANK() over (order by Score) as Rank,
DENSE_RANK() over(order by Score) as rownum
from level_details2
select top 3 MyUnknownColumn,Score,Difficulty,
RANK() over (partition by MyUnknownColumn order by Score) as DENCERank,
ROW_NUMBER() over (partition by MyUnknownColumn order by Score) as rownum
from level_details2
```

Dev_ID	Score	Difficulty	Rank	DENCERank	rownum
bd_013	100	Difficult	1	1	1
bd_013	100	Difficult	1	1	2
bd_013	540	Low	3	2	3

QUERIES AND OUTPUT

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

```
select l.DEV_ID, MIN(l.start_datetime) AS first_login_datetime
from level_details2 l
JOIN player_details p ON p.P_ID = l.P_ID
GROUP BY l.Dev_ID
```

Dev_ID	first_login_datetime
bd_013	2022-10-11 02:23:45.000
bd_015	2022-10-11 18:45:55.000
bd_017	2022-10-12 07:30:18.000
rf_013	2022-10-11 05:20:40.000
rf_015	2022-10-11 19:34:25.000
rf_017	2022-10-11 09:28:56.000
wd_019	2022-10-12 23:19:17.000
zm_013	2022-10-11 13:00:22.000
zm_015	2022-10-11 14:05:08.000
zm_017	2022-10-11 14:33:27.000

QUERIES AND OUTPUT

9. FIND THE TOP 5 SCORES BASED ON EACH DIFFICULTY LEVEL AND RANK THEM IN INCREASING ORDER USING `RANK`. DISPLAY `DEV_ID` AS WELL.

```
select top 5 Dev_ID,Score,Difficulty,
RANK() over (partition by Difficulty order by Score)as Rank,
DENSE_RANK() over (partition by Difficulty order by Score) as DENCERank,
ROW_NUMBER() over (partition by Difficulty order by Score) as rownum
from level_details2
```

Dev_ID	Score	Difficulty	Rank	DENCERank	rownum
bd_013	100	Difficult	1	1	1
zm_017	100	Difficult	1	1	2
bd_013	100	Difficult	1	1	3
wd_019	100	Difficult	1	1	4
rf_013	235	Difficult	5	2	5

QUERIES AND OUTPUT

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.P_ID,p.PNAME,l.Dev_ID, MIN(l.start_datetime)over(partition by p.P_ID) AS
first_login_datetime
from player_details p
join level_details2 l on p.P_ID=l.P_ID
```

P_ID	PName	Dev_ID	first_login_datetime
211	breezy-indigo-starfish	bd_017	2022-10-12 13:23:45.000
211	breezy-indigo-starfish	bd_013	2022-10-12 13:23:45.000
211	breezy-indigo-starfish	rf_013	2022-10-12 13:23:45.000
211	breezy-indigo-starfish	zm_015	2022-10-12 13:23:45.000
211	breezy-indigo-starfish	zm_017	2022-10-12 13:23:45.000
211	breezy-indigo-starfish	rf_017	2022-10-12 13:23:45.000
224	nippy-peach-neanderthal	rf_017	2022-10-14 01:15:56.000
224	nippy-peach-neanderthal	bd_015	2022-10-14 01:15:56.000
224	nippy-peach-neanderthal	bd_013	2022-10-14 01:15:56.000
224	nippy-peach-neanderthal	bd_013	2022-10-14 01:15:56.000
242	slaphappy-cinnamon-s...	bd_013	2022-10-13 01:14:29.000
242	slaphappy-cinnamon-s...	zm_015	2022-10-13 01:14:29.000

QUERIES AND OUTPUT

11. FOR EACH PLAYER AND DATE, DETERMINE HOW MANY `KILL_COUNTS` WERE PLAYED BY THE PLAYER SO FAR.
A) USING WINDOW FUNCTIONS
B) WITHOUT WINDOW FUNCTIONS

```
select P.PName,L.start_datetime,
SUM(L.kill_count) OVER (PARTITION BY P.P_ID ORDER BY L,start_datetime) AS TOTAL_Games,
DENSE_RANK() OVER (PARTITION BY P.P_ID ORDER BY L,start_datetime) AS Rank
FROM player_details P
JOIN level_details2 L ON P.P_ID = L.P_ID
```

```
select P.PName,L.start_datetime,sum(L.kill_count) AS TOTAL_Games
from player_details p
join level_details2 l
on p.P_ID=l.P_ID
group by P.PName,L.start_datetime
order y TOTAL_Games
```

	PName	start_datetime	TOTAL_Games	Rank
1	breezy-indigo-starfish	2022-10-12 13:23:45.000	20	1
2	breezy-indigo-starfish	2022-10-12 18:30:30.000	45	2
3	breezy-indigo-starfish	2022-10-13 05:36:15.000	75	3
4	breezy-indigo-starfish	2022-10-13 22:30:18.000	89	4
5	breezy-indigo-starfish	2022-10-14 08:56:24.000	98	5
6	breezy-indigo-starfish	2022-10-15 11:41:19.000	113	6
7	nippy-peach-neanderthal	2022-10-14 01:15:56.000	20	1
8	nippy-peach-neanderthal	2022-10-14 08:21:49.000	54	2
9	nippy-peach-neanderthal	2022-10-15 05:30:28.000	84	3
10	nippy-peach-neanderthal	2022-10-15 13:43:50.000	112	4

	PName	start_datetime	TOTAL_Games
1	homey-alizarin-gar	2022-10-13 04:04:04.000	2
2	sloppy-denim-wolfhound	2022-10-14 07:32:00.000	3
3	skinny-grey-quetzal	2022-10-14 18:23:29.000	3
4	silly-taupe-ray	2022-10-14 19:35:49.000	4
5	skinny-grey-quetzal	2022-10-14 05:05:05.000	4
6	homey-alizarin-gar	2022-10-14 01:25:30.000	4
7	fuzzy-cornflower-whippet	2022-10-15 06:30:20.000	4
8	ugly-goldenrod-numbat	2022-10-15 10:19:30.000	4
9	dorky-heliotrope-barracuda	2022-10-13 06:30:20.000	4
10	lanky-asparagus-gar	2022-10-12 01:45:17.000	4
11	chummy-flax-crab	2022-10-12 14:20:40.000	5
12	leaky-magnolia-iguana	2022-10-15 18:00:00.000	5
13	silly-taupe-ray	2022-10-14 15:15:15.000	7

QUERIES AND OUTPUT

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

```
SELECT start_datetime, SUM(Stages_crossed) OVER (ORDER By start_datetime) AS
cumulative_stages
From Level_deatails2
```

start_datetime	cumulative_stages
2022-10-11 02:23:45.000	4
2022-10-11 05:20:40.000	11
2022-10-11 09:28:56.000	13
2022-10-11 13:00:22.000	20
2022-10-11 14:05:08.000	23
2022-10-11 14:33:27.000	33
2022-10-11 15:15:15.000	40
2022-10-11 17:47:09.000	50
2022-10-11 18:45:55.000	53
2022-10-11 19:19:19.000	58
2022-10-11 19:28:43.000	64

QUERIES AND OUTPUT

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

```
WITH RankedLevels AS (
select*,
ROW_NUMBER() OVER (PARTITION BY P_ID ORDER BY satrt_datetime DESC) AS rn FROM
level_details2
)
SELECT P_ID, start_datetime,
SUM(Stages_crossed) OVER (PARTITION BY P_ID ORDER BY start_datetime ROWS BETWEEN
UNBOUNDER PROCEDING AND ! PRECEING) AS total_stages
FROM RankedLevels
Where rn > 1
```

P_ID	start_datetime	total_stages
211	2022-10-12 13:23:45.000	NULL
211	2022-10-12 18:30:30.000	4
211	2022-10-13 05:36:15.000	9
211	2022-10-13 22:30:18.000	14
211	2022-10-14 08:56:24.000	19
224	2022-10-14 01:15:56.000	NULL
224	2022-10-14 08:21:49.000	7
224	2022-10-15 05:30:28.000	12
242	2022-10-13 01:14:29.000	NULL

QUERIES AND OUTPUT

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

```
select top 3 Dev_ID,P_ID,SUM(Score) as score
from level_details2
group by Dev_ID,P_ID
order by score desc
```

	Dev_ID	P_ID	score
1	bd_013	224	9870
2	zm_017	683	8900
3	zm_017	632	5600

QUERIES AND OUTPUT

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.

```
WITH PlayerScores AS(
SELECT P_ID, SUM(Score) AS Total_Score
FROM level_details2
GROUP BY P_ID
),
AvgScores AS(
SELECT AVG(Total_Score) AS Avg_Score
FROM PlayerScores)
SELECT ps.P_ID,ps.Total_Score, a.Avg_Score
FROM PlayerScores ps, AvgScores a
WHERE ps.Total_Score > 0.5* a.Avg_Score
```

	P_ID	Total_Score	Avg_Score
1	211	10940	7040
2	224	16310	7040
3	242	6310	7040
4	300	4860	7040
5	310	13810	7040
6	368	8710	7040
7	429	13220	7040
8	483	17230	7040
9	590	8000	7040
10	632	10750	7040

SUMMARY

1. PLAYER DETAILS AT LEVEL 0: RETRIEVED PLAYER DETAILS, INCLUDING PLAYER ID, DEVICE ID, PLAYER NAME, AND DIFFICULTY LEVEL, FOR PLAYERS AT LEVEL 0.
2. AVERAGE KILL COUNT BY LEVEL 1 CODE: CALCULATED THE AVERAGE KILL COUNT FOR PLAYERS WITH SPECIFIC CONDITIONS ON EXTRA LIVES AND STAGES CROSSED, GROUPED BY LEVEL 1 CODE.
3. TOTAL STAGES CROSSED BY DIFFICULTY LEVEL: DETERMINED THE TOTAL NUMBER OF STAGES CROSSED AT EACH DIFFICULTY LEVEL FOR LEVEL 2 PLAYERS USING 'ZM_SERIES' DEVICES, SORTED IN DESCENDING ORDER OF STAGES CROSSED.
4. UNIQUE DATES PLAYED: EXTRACTED PLAYER ID AND THE TOTAL NUMBER OF UNIQUE DATES ON WHICH PLAYERS PLAYED GAMES.
5. LEVELWISE SUM OF KILL COUNTS: CALCULATED THE SUM OF KILL COUNTS FOR EACH LEVEL WHERE THE KILL COUNT EXCEEDED THE AVERAGE FOR MEDIUM DIFFICULTY.
6. SUM OF LIVES EARNED BY LEVEL AND LEVEL CODE: FOUND THE SUM OF LIVES EARNED FOR EACH LEVEL AND CORRESPONDING LEVEL CODE, EXCLUDING LEVEL 0.
7. TOP 3 SCORES BY DEVICE ID: IDENTIFIED THE TOP 3 SCORES FOR EACH DEVICE ID AND RANKED THEM USING ROW_NUMBER, INCLUDING THE DIFFICULTY LEVEL.
8. FIRST LOGIN DATE FOR EACH DEVICE ID: RETRIEVED THE FIRST LOGIN DATETIME FOR EACH DEVICE ID.
9. TOP 5 SCORES BY DIFFICULTY LEVEL: FOUND THE TOP 5 SCORES FOR EACH DIFFICULTY LEVEL AND RANKED THEM USING RANK, INCLUDING THE DEVICE ID.

10. FIRST LOGIN DEVICE ID FOR EACH PLAYER ID: DETERMINED THE DEVICE ID THAT WAS FIRST LOGGED IN BY EACH PLAYER.
11. CUMULATIVE KILL COUNTS BY PLAYER AND DATE: CALCULATED THE CUMULATIVE SUM OF KILL COUNTS FOR EACH PLAYER ON EACH DATE, WITH AND WITHOUT WINDOW FUNCTIONS.
12. CUMULATIVE STAGES CROSSED BY PLAYER ID: FOUND THE CUMULATIVE SUM OF STAGES CROSSED OVER START DATETIME FOR EACH PLAYER ID, EXCLUDING THE MOST RECENT START DATETIME.
13. TOP 3 SUM OF SCORES BY DEVICE ID: EXTRACTED THE TOP 3 HIGHEST SUMS OF SCORES FOR EACH DEVICE ID ALONG WITH THE CORRESPONDING PLAYER ID.
14. PLAYERS WITH SCORES ABOVE 50% OF AVERAGE SCORE: IDENTIFIED PLAYERS WHO SCORED MORE THAN 50% OF THE AVERAGE SCORE FOR EACH PLAYER ID.
15. STORED PROCEDURE FOR TOP HEADSHOTS COUNT: CREATED A STORED PROCEDURE TO FIND THE TOP 'N' HEADSHOTS COUNT FOR EACH DEVICE ID, RANKING THEM USING ROW_NUMBER AND INCLUDING THE DIFFICULTY LEVEL.

CONCLUSION

THE ANALYSIS PROVIDED VALUABLE INSIGHTS INTO PLAYER BEHAVIOR, GAME PERFORMANCE, AND DEVICE USAGE. THESE FINDINGS CAN BE UTILIZED TO OPTIMIZE GAME DEVELOPMENT, MARKETING STRATEGIES, AND PLAYER ENGAGEMENT INITIATIVES IN THE GAMING INDUSTRY.

THANK YOU