DECODE GAME ANALYSIS USING SQL

Internship Project

by Mentorness

By

Mounika S

On April 23, 2024

CONTENTS

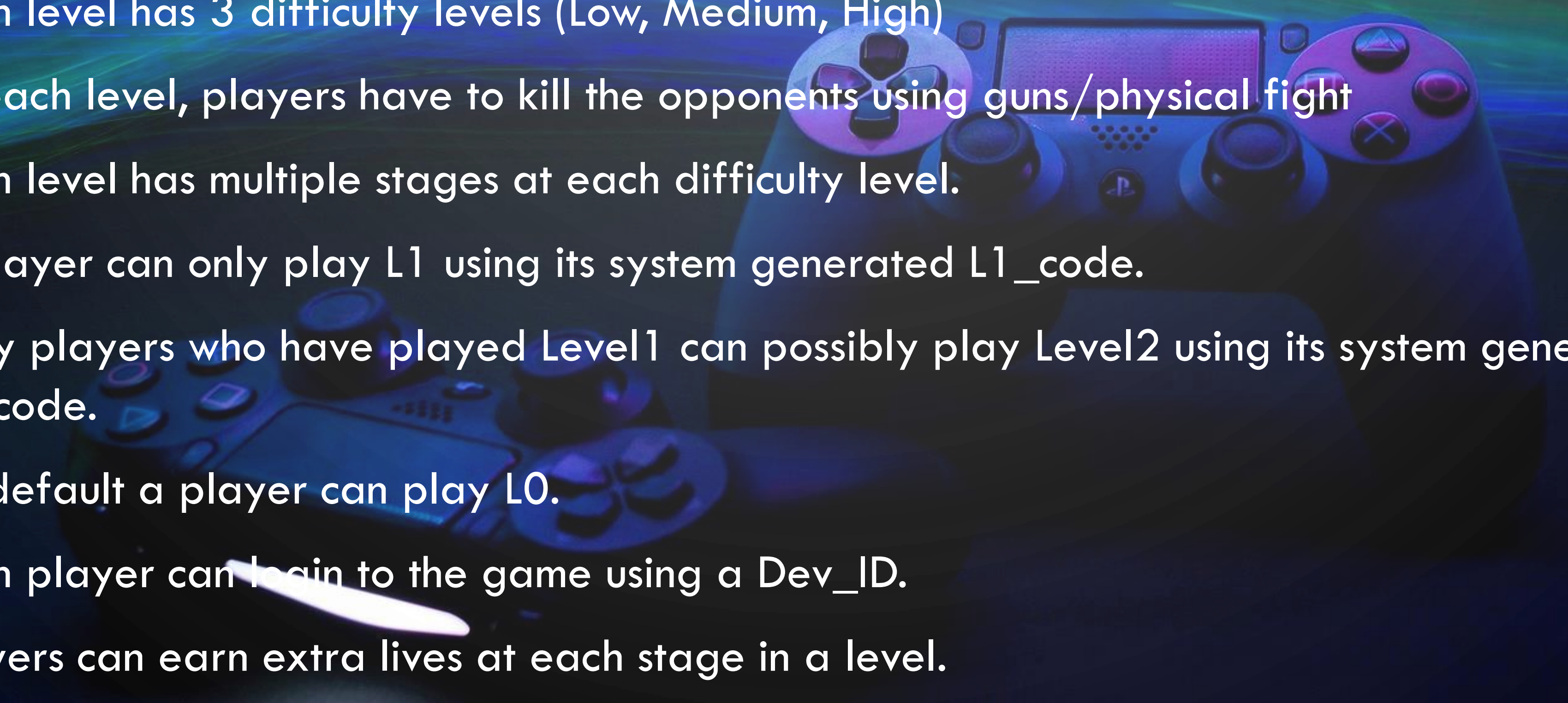
OVER VIEW OF THE PROJECT

Decode Game analysis project involves analyzing gaming behavior using a dataset provided by Mentorness.

We have to answer 17 questions. Extract the data from datasets and we have to analyze data by writing the queries and performing task from questions.

In this, I am using PostgreSQL, it is advanced relational database management system.

PROBLEM STATEMENT



- Project Name: Decode Gaming Behaviour
- 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 Level1 can possibly play Level2 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.

TWO TABLES ARE

- **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 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

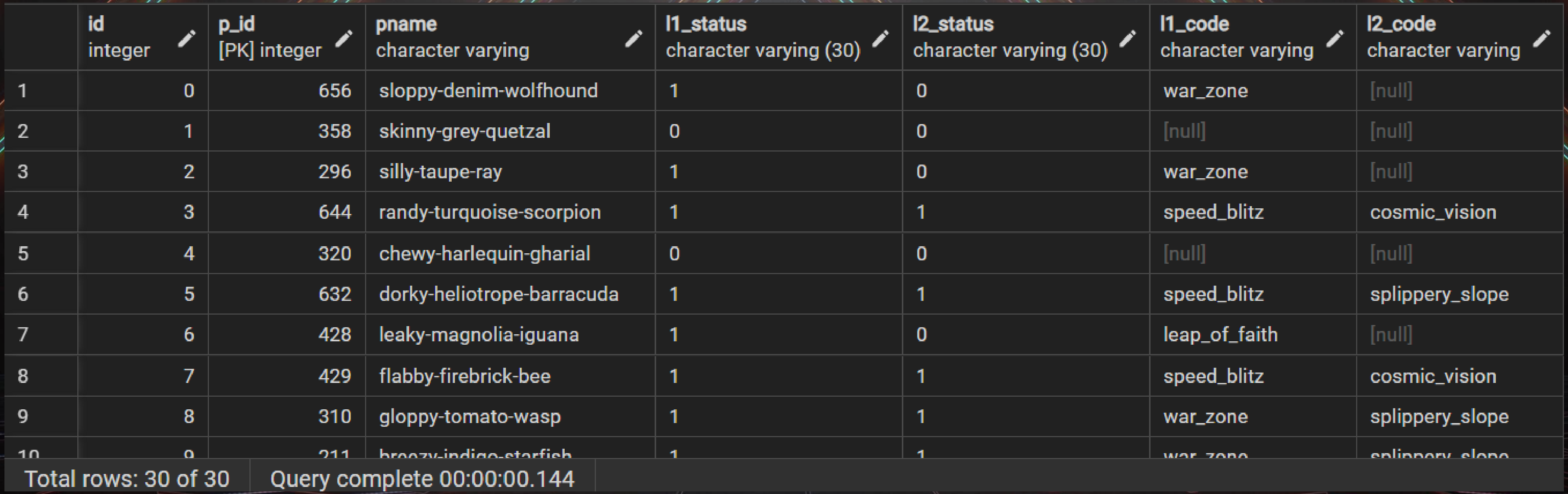
`headshots_count`: Headshots Count

`score`: Player Score

`lives_earned`: Extra Lives Earned

PLAYER DETAILS TABLE

SELECT * FROM player_details;

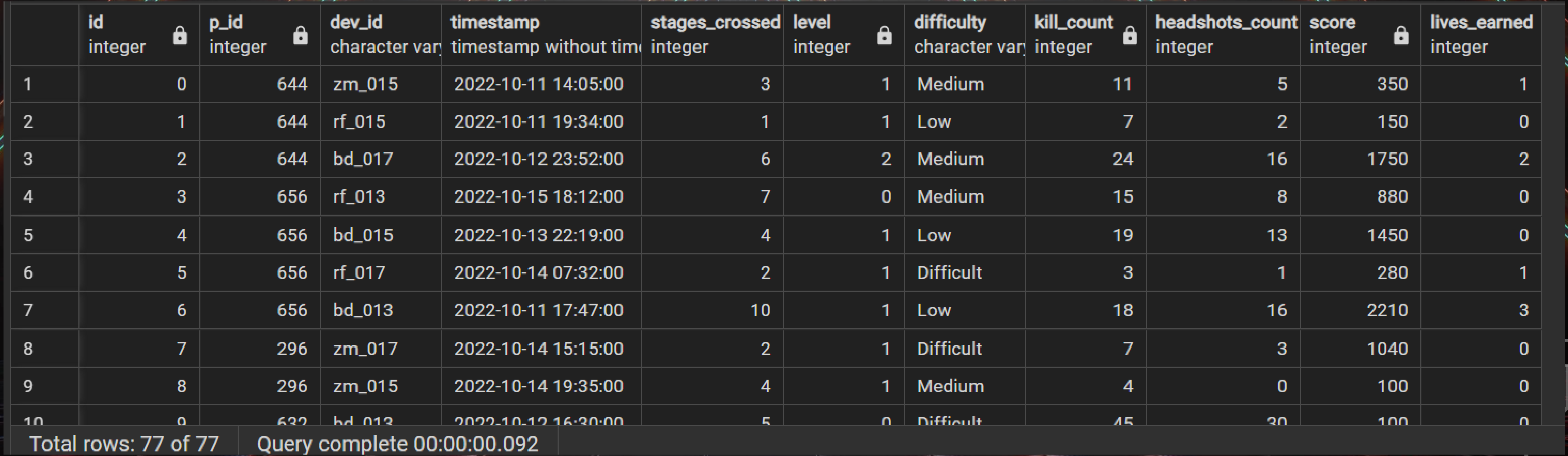


	id integer	p_id [PK] integer	pname character varying	l1_status character varying (30)	l2_status character varying (30)	l1_code character varying	l2_code character varying
1	0	656	sloppy-denim-wolfhound	1	0	war_zone	[null]
2	1	358	skinny-grey-quetzal	0	0	[null]	[null]
3	2	296	silly-taupe-ray	1	0	war_zone	[null]
4	3	644	randy-turquoise-scorpion	1	1	speed_blitz	cosmic_vision
5	4	320	chewy-harlequin-gharial	0	0	[null]	[null]
6	5	632	dorky-heliotrope-barracuda	1	1	speed_blitz	splippery_slope
7	6	428	leaky-magnolia-iguana	1	0	leap_of_faith	[null]
8	7	429	flabby-firebrick-bee	1	1	speed_blitz	cosmic_vision
9	8	310	gloppy-tomato-wasp	1	1	war_zone	splippery_slope
10	9	211	breezy-indigo-starfish	1	1	war_zone	splippery_slope

Total rows: 30 of 30   Query complete 00:00:00.144

LEVEL DETAILS TABLE

```
SELECT * FROM level_details;
```

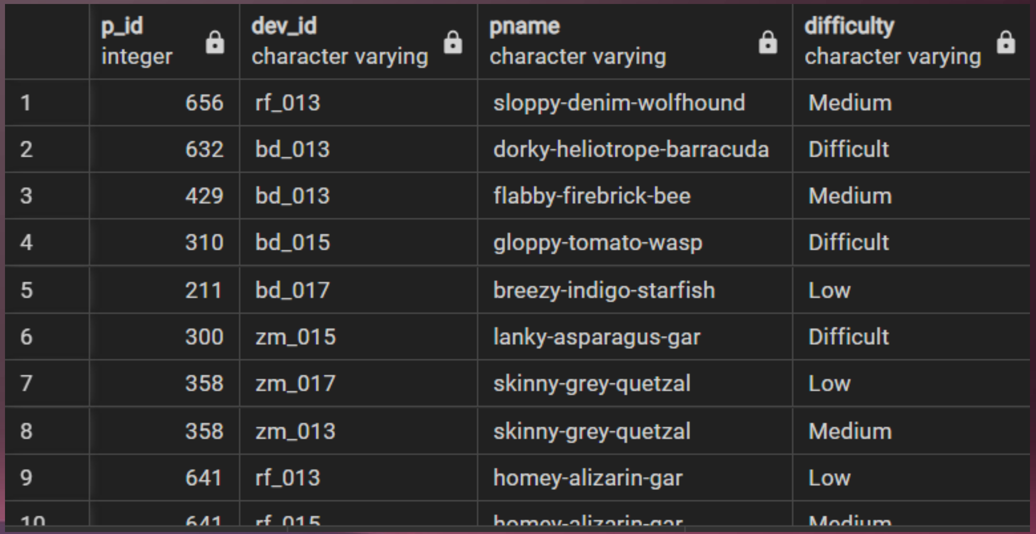


	id integer	p_id integer	dev_id character vary	timestamp timestamp without time	stages_crossed integer	level integer	difficulty character vary	kill_count integer	headshots_count integer	score integer	lives_earned integer
1	0	644	zm_015	2022-10-11 14:05:00	3	1	Medium	11	5	350	1
2	1	644	rf_015	2022-10-11 19:34:00	1	1	Low	7	2	150	0
3	2	644	bd_017	2022-10-12 23:52:00	6	2	Medium	24	16	1750	2
4	3	656	rf_013	2022-10-15 18:12:00	7	0	Medium	15	8	880	0
5	4	656	bd_015	2022-10-13 22:19:00	4	1	Low	19	13	1450	0
6	5	656	rf_017	2022-10-14 07:32:00	2	1	Difficult	3	1	280	1
7	6	656	bd_013	2022-10-11 17:47:00	10	1	Low	18	16	2210	3
8	7	296	zm_017	2022-10-14 15:15:00	2	1	Difficult	7	3	1040	0
9	8	296	zm_015	2022-10-14 19:35:00	4	1	Medium	4	0	100	0
10	9	632	bd_013	2022-10-12 16:30:00	5	0	Difficult	45	30	100	0

Total rows: 77 of 77 Query complete 00:00:00.092

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

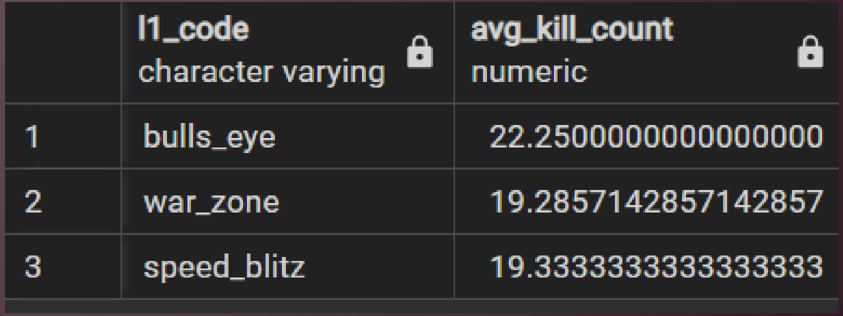
```
select pd.P_ID, ld.Dev_ID, pd.PName,
ld.Difficulty

from player_details as pd

inner join level_details as ld

on pd.P_ID = ld.P_ID

where ld.level = 0;
```



	p_id integer	dev_id character varying	pname character varying	difficulty character varying
1	656	rf_013	sloppy-denim-wolfhound	Medium
2	632	bd_013	dorky-heliotrope-barracuda	Difficult
3	429	bd_013	flabby-firebrick-bee	Medium
4	310	bd_015	gloppy-tomato-wasp	Difficult
5	211	bd_017	breezy-indigo-starfish	Low
6	300	zm_015	lanky-asparagus-gar	Difficult
7	358	zm_017	skinny-grey-quetzal	Low
8	358	zm_013	skinny-grey-quetzal	Medium
9	641	rf_013	homey-alizarin-gar	Low
10	641	rf_015	homey-alizarin-gar	Medium

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

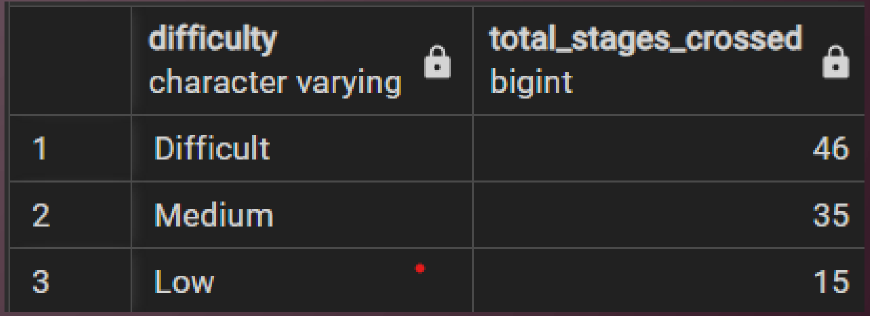
```
select pd.L1_Code, avg(ld.kill_count) as
Avg_Kill_Count

from player_details as pd

join level_details as 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 character varying	avg_kill_count numeric
1	bulls_eye	22.2500000000000000
2	war_zone	19.2857142857142857
3	speed_blitz	19.3333333333333333

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.

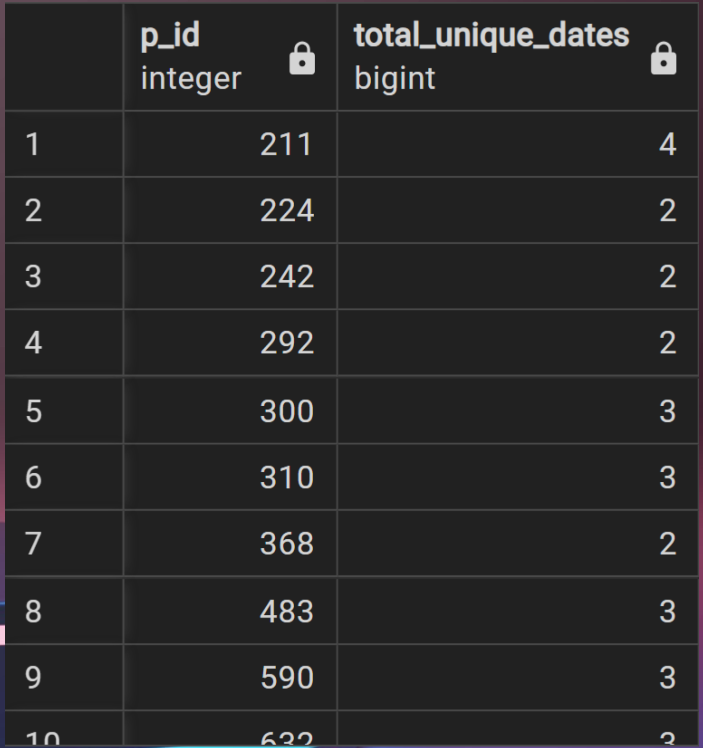
```
select difficulty, sum(stages_crossed) as
total_stages_crossed

from level_details

where level = 2 and Dev_ID like 'zm%'

group by Difficulty

order by total_stages_crossed desc;
```



	difficulty character varying	total_stages_crossed bigint
1	Difficult	46
2	Medium	35
3	Low	15

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

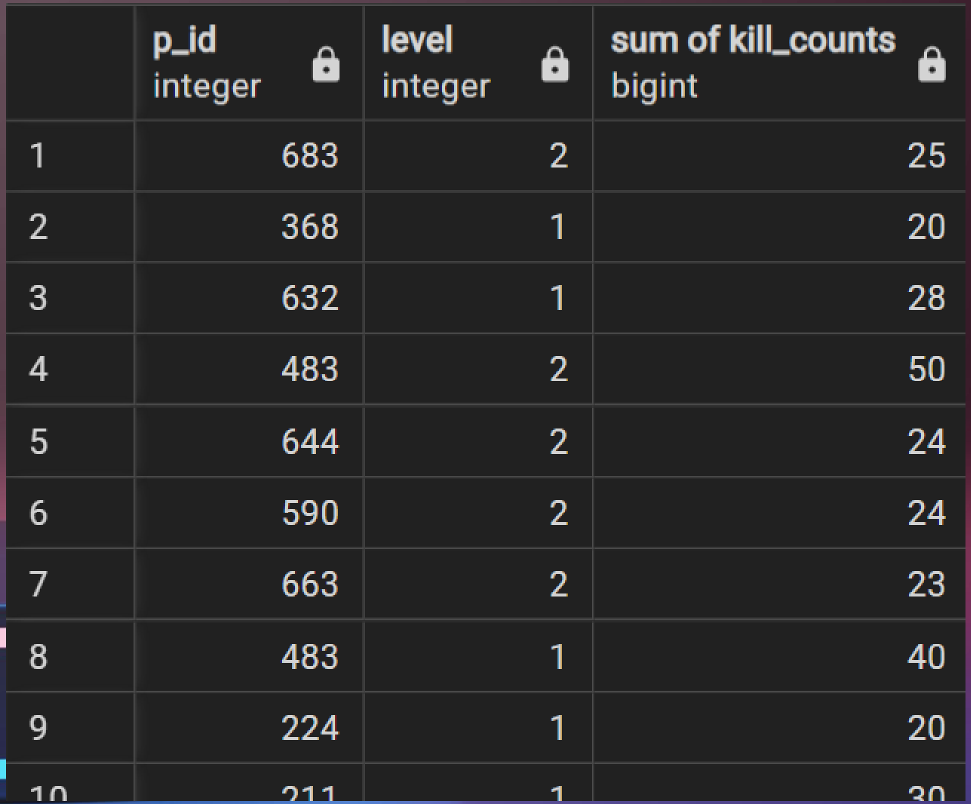
```
select P_ID, count(distinct date(timestamp)) as
total_unique_dates

from level_details

group by P_ID

having count(distinct date(timestamp)) > 1;
```



	p_id integer	total_unique_dates bigint
1	211	4
2	224	2
3	242	2
4	292	2
5	300	3
6	310	3
7	368	2
8	483	3
9	590	3
10	632	3

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

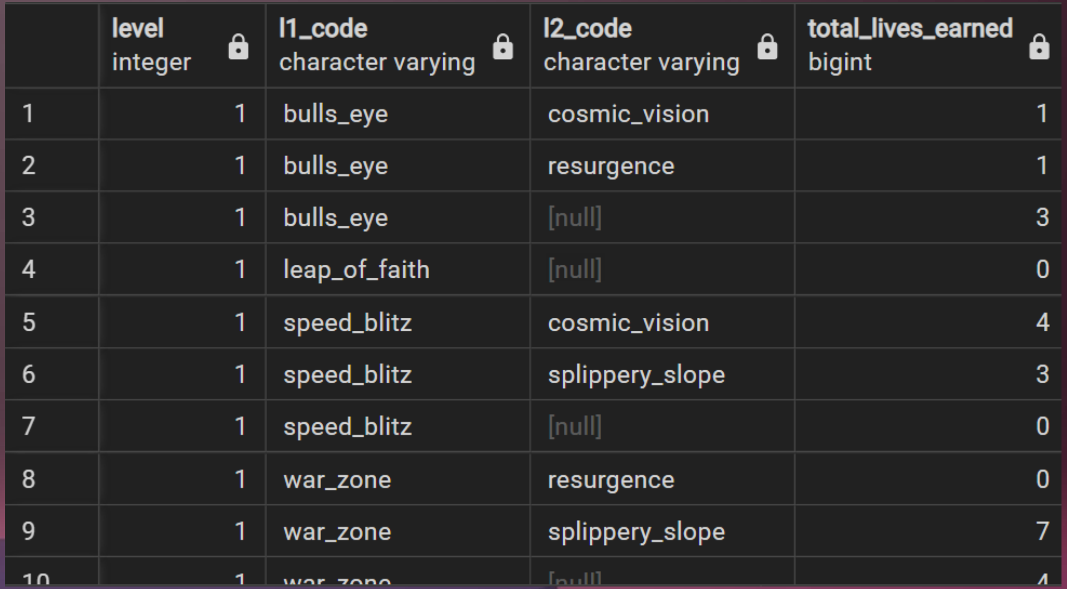
```
select P_ID, level, sum(Kill_Count) as "sum of
kill_counts"

from level_details

where Difficulty = 'Medium'

group by P_ID, level

having sum(Kill_Count) > (select
avg(Kill_Count) from level_details where
Difficulty = 'Medium');
```



	p_id integer	level integer	sum of kill_counts bigint
1	683	2	25
2	368	1	20
3	632	1	28
4	483	2	50
5	644	2	24
6	590	2	24
7	663	2	23
8	483	1	40
9	224	1	20
10	211	1	30

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

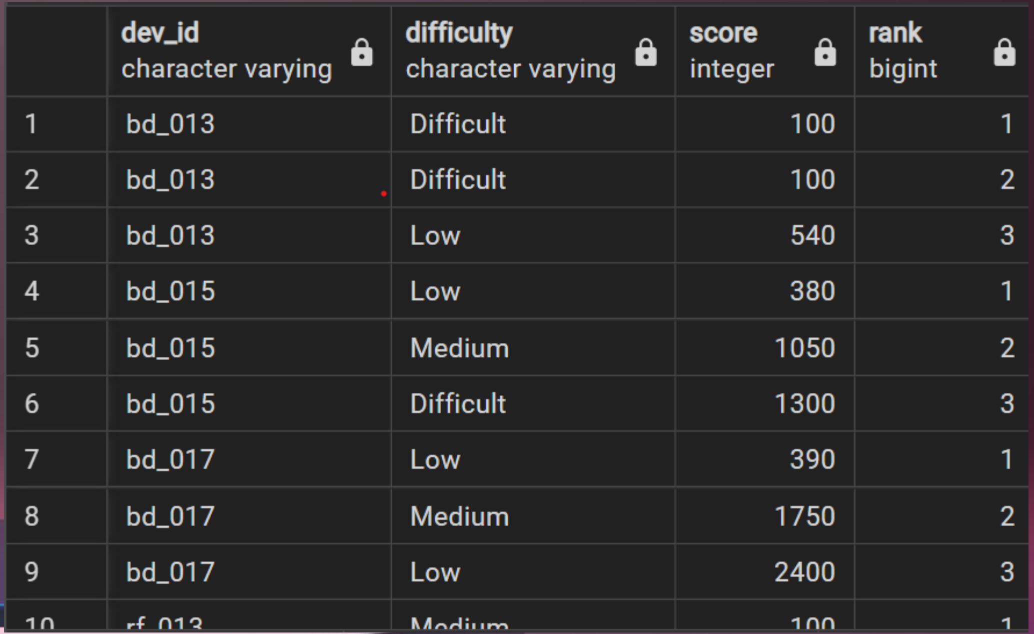
```
select ld.level, pd.L1_Code, pd.L2_Code,
sum(ld.Lives_Earned) as total_lives_earned

from level_details as ld

inner join player_details as pd

on ld.P_ID = pd.P_ID

where ld.level != 0

group by ld.level,pd.L1_Code, pd.L2_Code

order by ld.level asc;
```



	level integer	l1_code character varying	l2_code character varying	total_lives_earned bigint
1	1	bulls_eye	cosmic_vision	1
2	1	bulls_eye	resurgence	1
3	1	bulls_eye	[null]	3
4	1	leap_of_faith	[null]	0
5	1	speed_blitz	cosmic_vision	4
6	1	speed_blitz	splippery_slope	3
7	1	speed_blitz	[null]	0
8	1	war_zone	resurgence	0
9	1	war_zone	splippery_slope	7
10	1	war_zone	[null]	4

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.

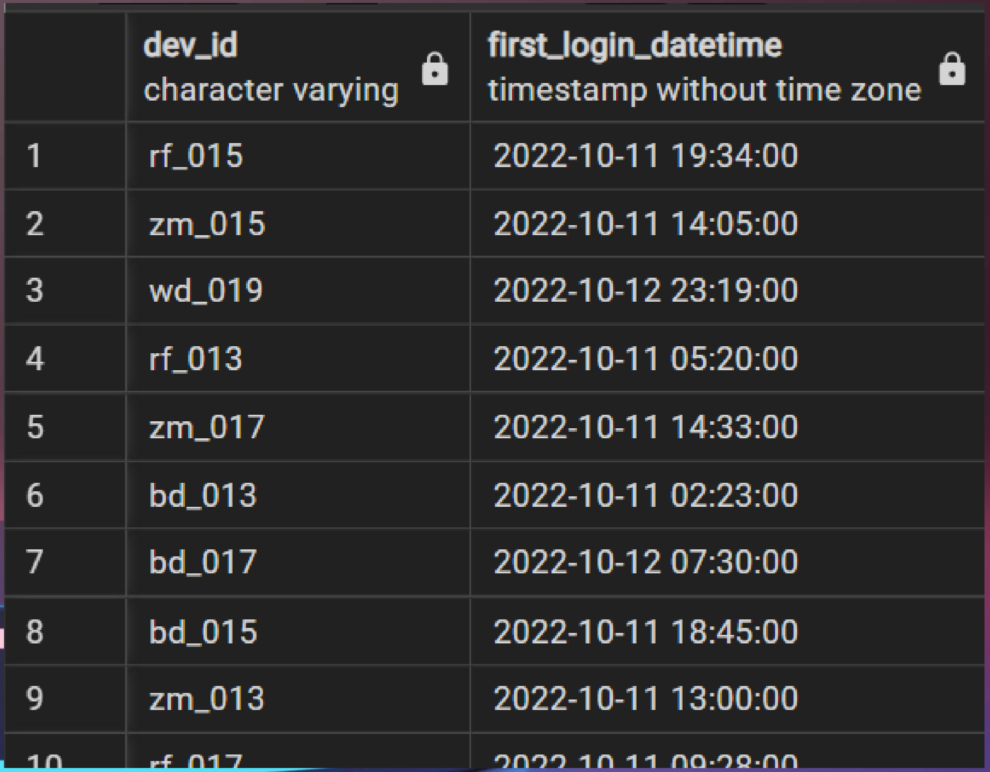
```
select dev_id, difficulty, score, rank

from (select score, dev_id, difficulty,

          row_number() over (partition by
dev_id order by score asc ) as rank

          from level_details ) as
ranked_scores

where rank <= 3;
```



	dev_id character varying	difficulty character varying	score integer	rank bigint
1	bd_013	Difficult	100	1
2	bd_013	Difficult	100	2
3	bd_013	Low	540	3
4	bd_015	Low	380	1
5	bd_015	Medium	1050	2
6	bd_015	Difficult	1300	3
7	bd_017	Low	390	1
8	bd_017	Medium	1750	2
9	bd_017	Low	2400	3
10	rf_013	Medium	100	1

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

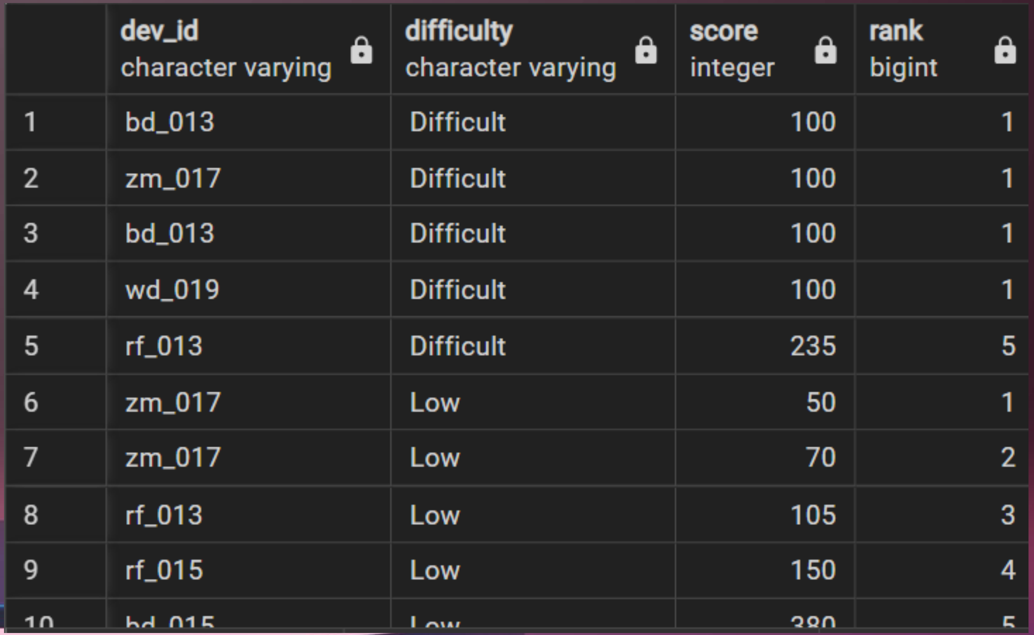
```
select dev_id, min(timestamp) as
first_login_datetime

from level_details

group by dev_id;
```



	dev_id character varying	first_login_datetime timestamp without time zone
1	rf_015	2022-10-11 19:34:00
2	zm_015	2022-10-11 14:05:00
3	wd_019	2022-10-12 23:19:00
4	rf_013	2022-10-11 05:20:00
5	zm_017	2022-10-11 14:33:00
6	bd_013	2022-10-11 02:23:00
7	bd_017	2022-10-12 07:30:00
8	bd_015	2022-10-11 18:45:00
9	zm_013	2022-10-11 13:00:00
10	rf_017	2022-10-11 09:28:00

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

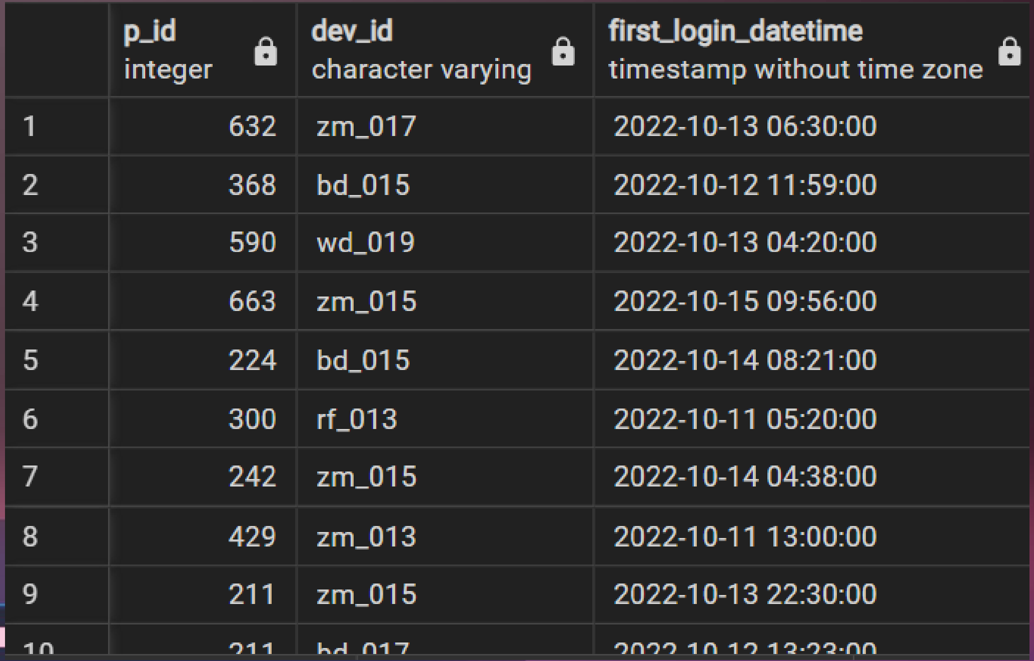
```
select dev_id, difficulty, score, rank

from (select score, dev_id, difficulty,

            rank() over (partition by difficulty
order by score asc ) as rank

            from level_details ) as
ranked_scores

where rank <= 5;
```



	dev_id character varying	difficulty character varying	score integer	rank bigint
1	bd_013	Difficult	100	1
2	zm_017	Difficult	100	1
3	bd_013	Difficult	100	1
4	wd_019	Difficult	100	1
5	rf_013	Difficult	235	5
6	zm_017	Low	50	1
7	zm_017	Low	70	2
8	rf_013	Low	105	3
9	rf_015	Low	150	4
10	bd_015	Low	380	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.

```
select p_id, dev_id, min(timestamp) as first_login_datetime

from level_details

group by dev_id, p_id;
```



	p_id integer	dev_id character varying	first_login_datetime timestamp without time zone
1	632	zm_017	2022-10-13 06:30:00
2	368	bd_015	2022-10-12 11:59:00
3	590	wd_019	2022-10-13 04:20:00
4	663	zm_015	2022-10-15 09:56:00
5	224	bd_015	2022-10-14 08:21:00
6	300	rf_013	2022-10-11 05:20:00
7	242	zm_015	2022-10-14 04:38:00
8	429	zm_013	2022-10-11 13:00:00
9	211	zm_015	2022-10-13 22:30:00
10	211	bd_017	2022-10-12 13:23:00

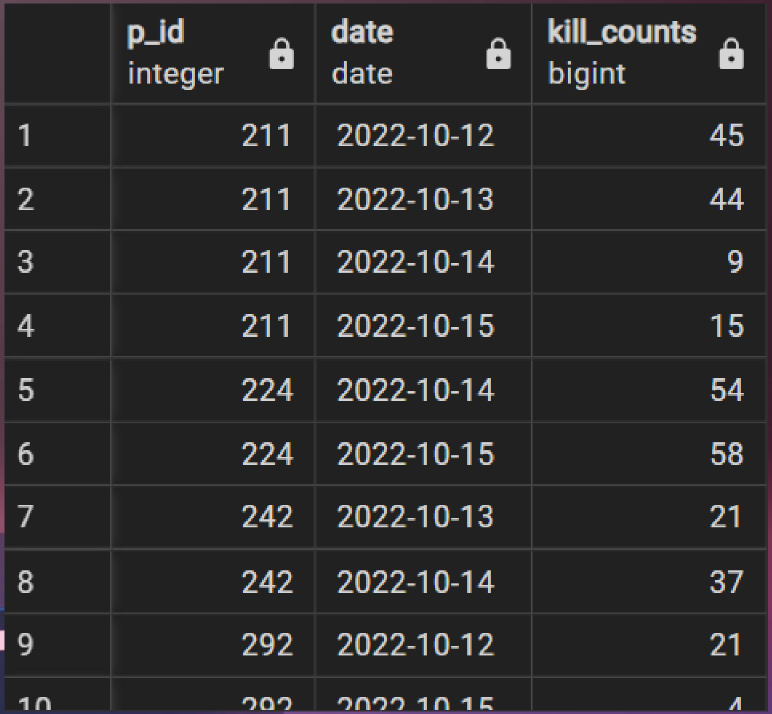
11. For each player and date, determine how many `kill_counts` were played by the player so far. That is, the total number of games played by the player until that date.

- a) Using window functions

```
select distinct p_id, date(timestamp) as date,
sum(kill_count) over(partition by p_id, date(timestamp) order by date(timestamp)) as "kill_counts"
from level_details
order by p_id, date(timestamp);
```

- b) Without window functions

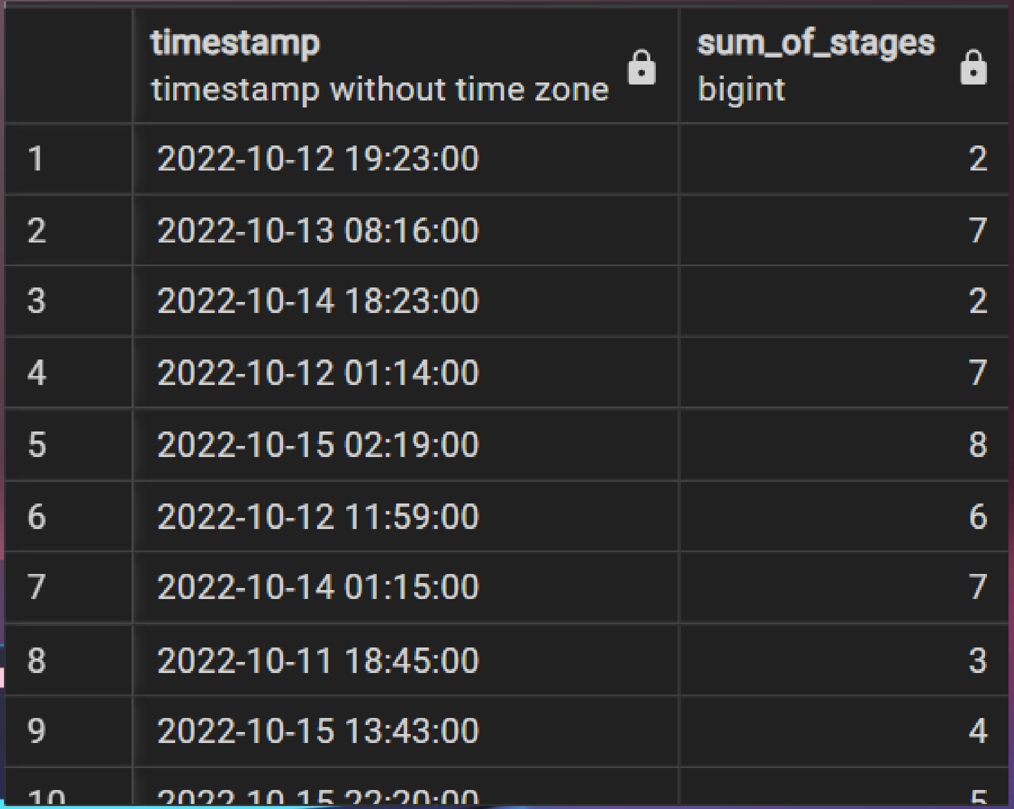
```
select p_id, date(timestamp), sum(kill_count) as "kill_counts"
from level_details
group by p_id, date(timestamp)
order by p_id, date(timestamp);
```



	p_id integer	date date	kill_counts bigint
1	211	2022-10-12	45
2	211	2022-10-13	44
3	211	2022-10-14	9
4	211	2022-10-15	15
5	224	2022-10-14	54
6	224	2022-10-15	58
7	242	2022-10-13	21
8	242	2022-10-14	37
9	292	2022-10-12	21
10	292	2022-10-15	4

12. Find the cumulative sum of stages crossed over `start_datetime`.

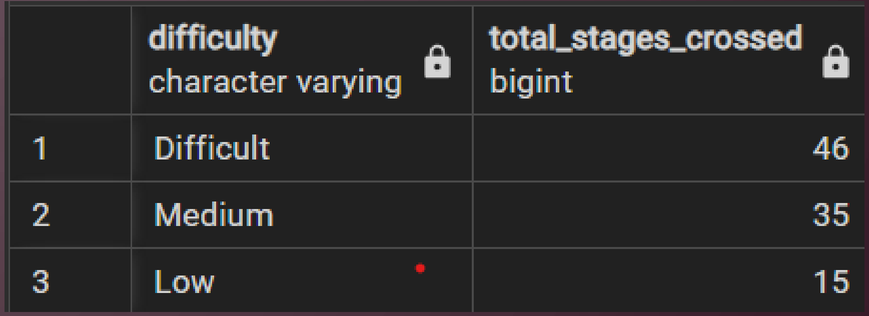
```
select timestamp, sum(Stages_crossed)
as sum_of_stages

from level_details

group by timestamp;
```



	timestamp timestamp without time zone	sum_of_stages bigint
1	2022-10-12 19:23:00	2
2	2022-10-13 08:16:00	7
3	2022-10-14 18:23:00	2
4	2022-10-12 01:14:00	7
5	2022-10-15 02:19:00	8
6	2022-10-12 11:59:00	6
7	2022-10-14 01:15:00	7
8	2022-10-11 18:45:00	3
9	2022-10-15 13:43:00	4
10	2022-10-15 22:20:00	5

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

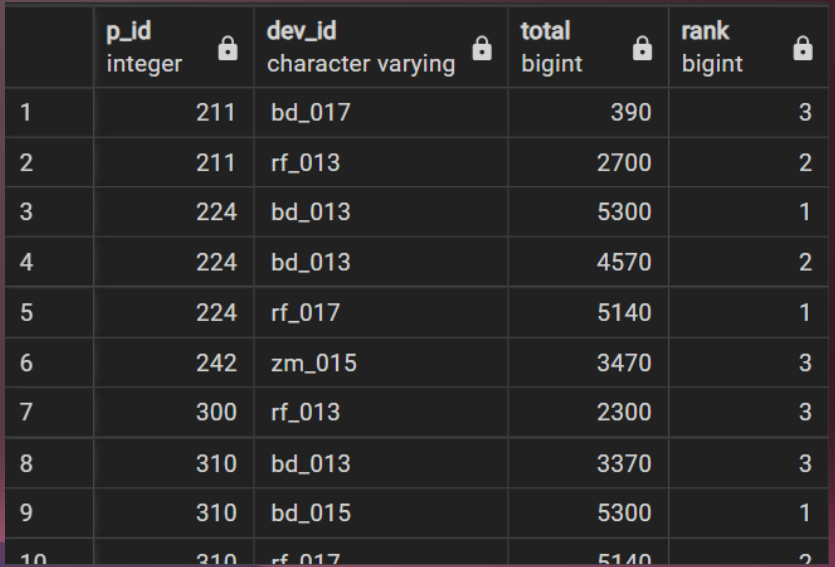
```
select p_id, TimeStamp, sum(Stages_crossed) as
sum_of_stages

from (select p_id, timestamp, stages_crossed,

            row_number() over (partition by p_id order
by TimeStamp desc) as rank from level_details )

where rank > 1

group by p_id, TimeStamp

order by p_id;
```



	difficulty character varying	total_stages_crossed bigint
1	Difficult	46
2	Medium	35
3	Low	15

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

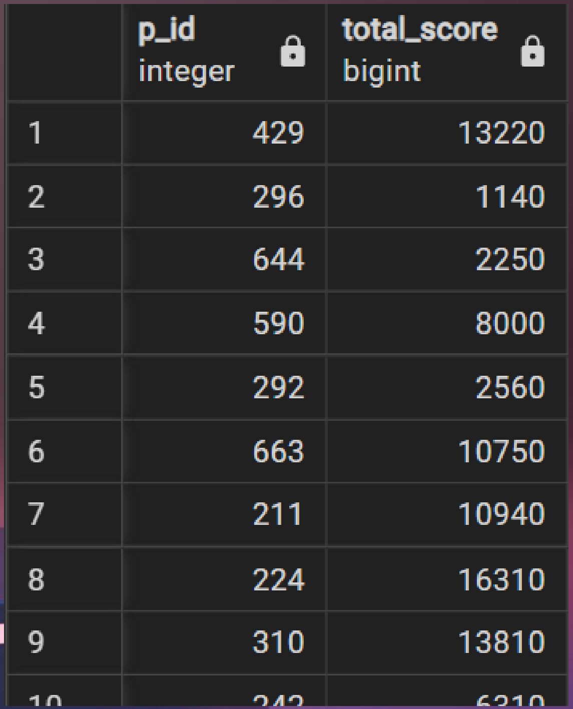
```
select P_id, dev_id, sum(score) as total, rank

from (select P_id, dev_id, score,

          row_number() over (partition by dev_id order by
sum(score) desc) as rank

          from level_details

          group by p_id, dev_id, score) as tb

where rank < 4

group by P_id, dev_id, rank

order by p_id;
```



	p_id integer	dev_id character varying	total bigint	rank bigint
1	211	bd_017	390	3
2	211	rf_013	2700	2
3	224	bd_013	5300	1
4	224	bd_013	4570	2
5	224	rf_017	5140	1
6	242	zm_015	3470	3
7	300	rf_013	2300	3
8	310	bd_013	3370	3
9	310	bd_015	5300	1
10	310	rf_017	5140	2

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

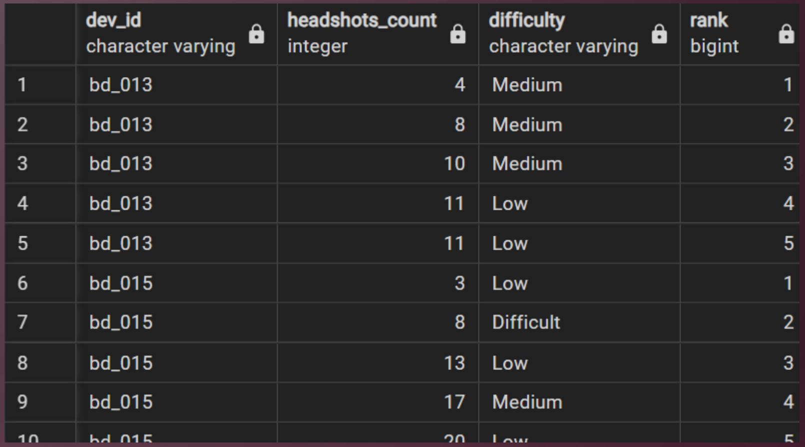
```
select p_id, sum(score) as total_score

from level_details

group by p_id

having sum(score)  > 0.5 * (select
avg(score) from level_details)
```



	p_id integer	total_score bigint
1	429	13220
2	296	1140
3	644	2250
4	590	8000
5	292	2560
6	663	10750
7	211	10940
8	224	16310
9	310	13810
10	242	6310

16. 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.

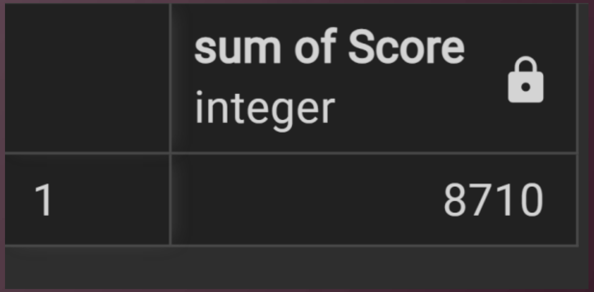
```
create or replace procedure public.top_n_headshots_procedure(IN n integer,
inout get_result refcursor) language 'plpgsql' as $body$
begin
            open get_result for
            select dev_id, Headshots_Count, difficulty, rank
            from (select dev_id, difficulty, Headshots_Count, row_number()
over (partition by dev_id order by Headshots_Count asc) as rank
                          from level_details
                         )as new_table
            where rank <= n
--group by dev_id, rank, difficulty, Headshots_Count
            order by dev_id, rank;
end
$body$;

call public.top_n_headshots_procedure (5, 'result');
fetch all in "result";
```



	dev_id character varying	headshots_count integer	difficulty character varying	rank bigint
1	bd_013	4	Medium	1
2	bd_013	8	Medium	2
3	bd_013	10	Medium	3
4	bd_013	11	Low	4
5	bd_013	11	Low	5
6	bd_015	3	Low	1
7	bd_015	8	Difficult	2
8	bd_015	13	Low	3
9	bd_015	17	Medium	4
10	bd_015	20	Low	5

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

```
create or replace function total_score_func (player_id int) returns int
as $body$
declare total_score int;
begin
        select sum(score) into total_score from level_details
        where p_id = player_id;
        return total_score;
        DROP FUNCTION total_score_func(integer);
end;
$body$ language plpgsql;
select total_score_func(368) as "sum of Score ";
```



	sum of Score integer
1	8710

INSIGHTS

Throughout this project, I gained valuable insights into various aspects of SQL query writing and data analysis. Some key learnings include:

- **window funtions**: it allowed me to calculate aggregate values, perform ranking, and generate cumulative sums with ease.
- **Stored procedure**: By creating stored procedures,I gained a deeper understanding of how to reuse sets of SQL statements and enabled me to execute complex tasks efficiently.
- **Create function:** By creating functions, I learned how to use functions and how to call where it is set of queries execute at last by calling the function name
- **Subqueries**: Writing subqueries helped me extract and manipulate data, and allowed me to nest queries within other queries, enabling me to filter, aggregate, and correlate data effectively.

These insights not only improved my SQL skills but also equipped me with valuable knowledge for future data analysis projects.

CONCLUSION

- The Decode Game Analysis project focuses on analyzing gaming behavior using SQL to extract actionable insights.
- Within the game, players encounter multiple levels, each presenting increasingly difficult challenges to overcome.
- The dataset contains vital information such as player IDs, device IDs, player names, and levels so on.
- By utilizing SQL queries, we can retrieve specific information and conduct various analyses to uncover patterns, trends, and relationships within the data.
- SQL queries play a crucial role in unlocking valuable insights hidden within the game datasets, empowering us to make informed decisions and optimizations for enhanced gameplay experiences.

THANK YOU