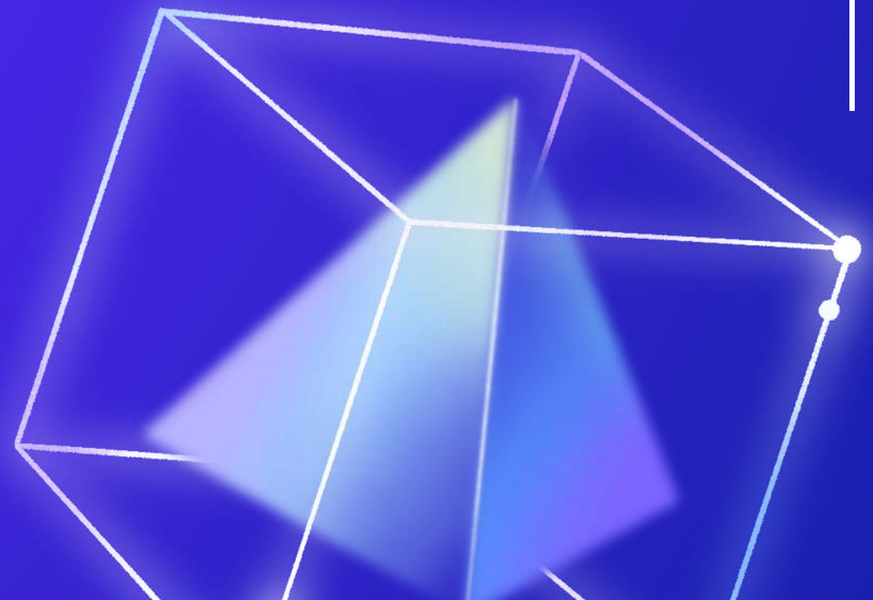
DECODE GAMING BEHAVIOUR

LEVERAGING DATA TO ENHANCE GAMING INSIGHTS

Submitted By:
Shweta Bhardwaj

TABLE OF CONTENTS



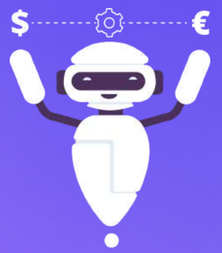
INTRODUCTION

In the gaming industry, data analysis is crucial for understanding player behavior, personalizing gaming experiences, improving game design, optimizing monetization strategies, and predicting trends. By leveraging data, developers can make informed decisions, enhance player satisfaction, and gain a competitive edge in the market.

PROJECT OBJECTIVES

OBJECTIVE 01

We will be working with a dataset related to a game. The dataset includes two tables: `Player Details` and `Level Details`.



OBJECTIVE 02

We have 15 questions for which we have to find the answers by writing SQL queries.

OBJECTIVE 03

We will be using MySQL Workbench for this after importing the datasets.

ANALYTICAL APPROACH

```sql
use game_analysis;
-- Q1) 'P_ID' , 'Dev_ID' , 'Difficulty_level' of all players at level 0
SELECT P_ID, Dev_ID, Difficulty
FROM level_details2
WHERE Level = 0;

-- Q2) Level1_code wise Avg_Kill_Count where lives_earned is 2 and atleast
--    3 stages are crossed
select 'L1_Code' , AVG(Kill_Count)
from player_details, level_details2
where Lives_Earned = 2
group by L1_Code
HAVING Count(distinct Stages_crossed) >= 3;
-- Q3) The total number of stages crossed at each diffuculty level
-- where for Level2 with players use zm_series devices. Arrange the result
-- in decsreasing order of total number of stages crossed.
select 'Difficulty' , SUM(Stages_crossed)
```

```sql
-- Q13) 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,
        ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY SUM(Score) DESC) AS Scorerank
    FROM level_details2
    GROUP BY Dev_ID, P_ID
)
SELECT
    Dev_ID,
    P_ID,
    total_score
FROM
    RankedScores
WHERE
    Scorerank <= 3;
```

```sql
-- Q14) players who scored more than 50% of the averag
-- scores for each `P_ID`.
    SELECT
    P_ID,
    SUM(Score) AS total_score
FROM
    level_details2
GROUP BY
    P_ID
HAVING
    SUM(Score) > 0.5 * (
        SELECT AVG(sum_score) FROM (
            SELECT
                SUM(Score) AS sum_score
            FROM
                level_details2
            GROUP BY
                P_ID
        ) AS avg_scores
    );
```

```sql
-- Q6) Level and its corresponding Level wise sum of
-- excluding level 0. Arrange in asecending order of
SELECT
    Level,
    L1_Code,
    L2_Code,
    SUM(Lives_Earned) AS total_lives_earned
from player_details , level_details2
WHERE
    Level != 0
GROUP BY
    Level, L1_code, L2_Code
ORDER BY
    Level ASC;
```

```sql
) Extract P_ID and the total number of unique dates for those players
o have played games on multiple days.
t 'P_ID' , Count(distinct date (TimeStamp))
player_details , level_details2
 by 'P_ID'
G Count(distinct date (TimeStamp)) > 1;
) P_ID and level wise sum of kill_counts where kill_count
 greater than avg kill count for the Medium difficulty.
T
_ID,
evel,
UM(Kill_Count) AS levelwise_sum_kill_counts
level_details2

ifficulty = 'Medium'
ND Kill_Count > (
    SELECT AVG(Kill_Count)
    FROM level_details2
    WHERE Difficulty = 'Medium'
)
GROUP BY
    P_ID, Level;
```

```sql
-- Q7) Top 3 score based on each dev_id and Rank them in increasing order
-- using Row_Number. Display difficulty as well.
WITH RankedScores AS (
    SELECT
        Dev_ID,
        Score,
        Difficulty,
        ROW_NUMBER() OVER (PARTITION BY Dev_ID ORDER BY Score ASC) AS ScoreRank
    FROM player_details , level_details2
    )
SELECT
    Dev_ID,
    Score,
    Difficulty
FROM
    RankedScores
WHERE
    ScoreRank <= 3;
```

OUTPUTS

Level	L1_Code	L2_Code	total_lives_earned
1	bulls_eye	resurgence	23
1	bulls_eye	splippery_slope	23
1	leap_of_faith		46
1	leap_of_faith	resurgence	23
1	speed_blitz		46
1	speed_blitz	cosmic_vision	69
1	speed_blitz	splippery_slope	46
1	war_zone		69
1	war_zone	resurgence	23
1	war_zone	splippery_slope	115
2			255
2	bulls_eye		153
2	bulls_eye	cosmic_vision	51
2	bulls_eye	resurgence	51
2	bulls_eye	splippery_slope	51

Dev_ID	MIN(TimeStamp)
zm_015	2022-10-11 14:05:08
rf_015	2022-10-
bd_017	2022-10-12 07:30:18
rf_013	2022-10-11 05:20:40
bd_015	2022-10-11 18:45:55
rf_017	2022-10-11 09:28:56
bd_013	2022-10-11 02:23:45
zm_017	2022-10-11 14:33:27
zm_013	2022-10-11 13:00:22
wd_019	2022-10-12 23:19:17

	dium	4	1
	edium	8	2
bd_013	Medium	10	3
bd_013	Difficult	11	4
bd_013	Low	11	5
bd_015	Low	3	1
bd_015	Difficult	8	2
bd_015	Low	13	3
bd_015	Medium	17	4
bd_015	Low	20	5
bd_017	Low	15	1
bd_017	Medium	16	2
bd_017	Low	18	3
rf_013	Low	3	1
rf_013	Medium	6	2

P_ID	Dev_ID	first_login_datetime
644	zm_015	2022-10-11 14:05:08
644	rf_015	2022-10-11 19:34:25
644	bd_017	2022-10-12 23:52:18
656	rf_013	2022-10-15 18:12:50
656	bd_015	2022-10-13 22:19:45
656	rf_017	2022-10-14 07:32:00
656	bd_013	2022-10-11 17:47:09
296	zm_017	2022-10-14 15:15:15
296	zm_015	2022-10-14 19:35:49
632	bd_013	2022-10-12 16:30:30
632	rf_013	2022-10-12 19:36:40
632	zm_017	2022-10-13 06:30:20
632	zm_015	2022-10-13 10:56:17
428	bd_015	2022-10-15 18:00:00
429	rf_017	2022-10-11 09:28:56

L1_Code	AVG(Kill_Count)
L1_Code	20.0000
L1_Code	20.0000
L1_Code	20.0000
L1_Code	20.0000
L1_Code	20.0000

ID	date	total_kill_counts_so_far
L	2022-10-12	20
L	2022-10-12	45
L	2022-10-13	75
L	2022-10-13	89
L	2022-10-14	98
L	2022-10-15	113
4	2022-10-14	20
4	2022-10-14	54
4	2022-10-15	84
4	2022-10-15	112
2	2022-10-13	21
242	2022-10-14	58
292	2022-10-12	21
292	2022-10-15	25
296	2022-10-14	7

AGGREGATE FUNCTIONS

(SUM , AVG , COUNT , MAX , MIN)

SUB QUERIES

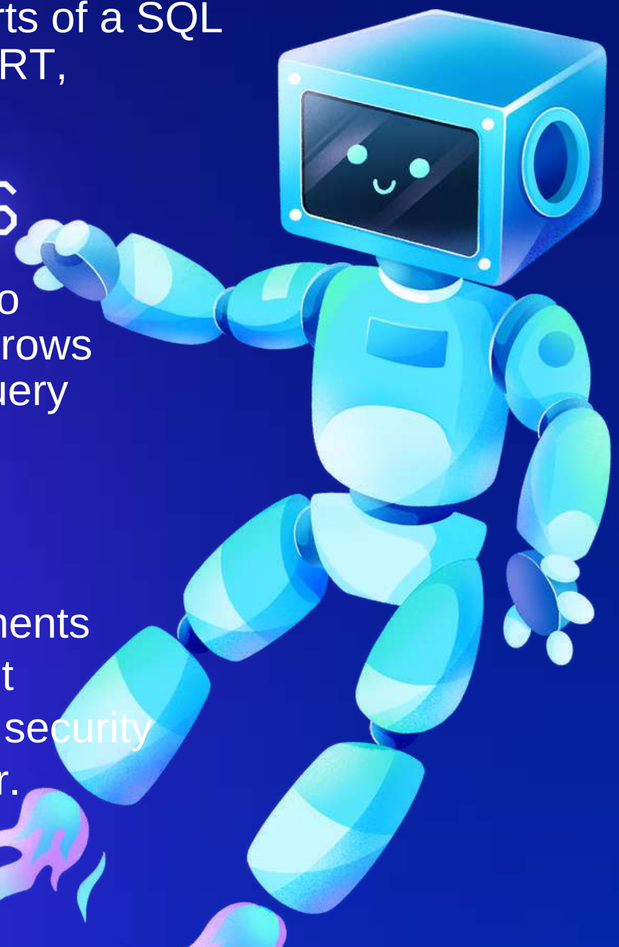
Subquery is a query nested within another SQL query. It can be used in various parts of a SQL statement, such as SELECT, INSERT, UPDATE, or DELETE statements.

WINDOW FUNCTIONS

Window functions in SQL are used to perform calculations across a set of rows related to the current row within a query result set.

STORED PROCEDURE

It is a precompiled collection of SQL statements that are stored and can be executed later. It enhance performance, maintainability, and security by centralizing logic on the database server.



THANK YOU!