

IPL Match Bidding App

TITLE :

Pie-in-the-Sky

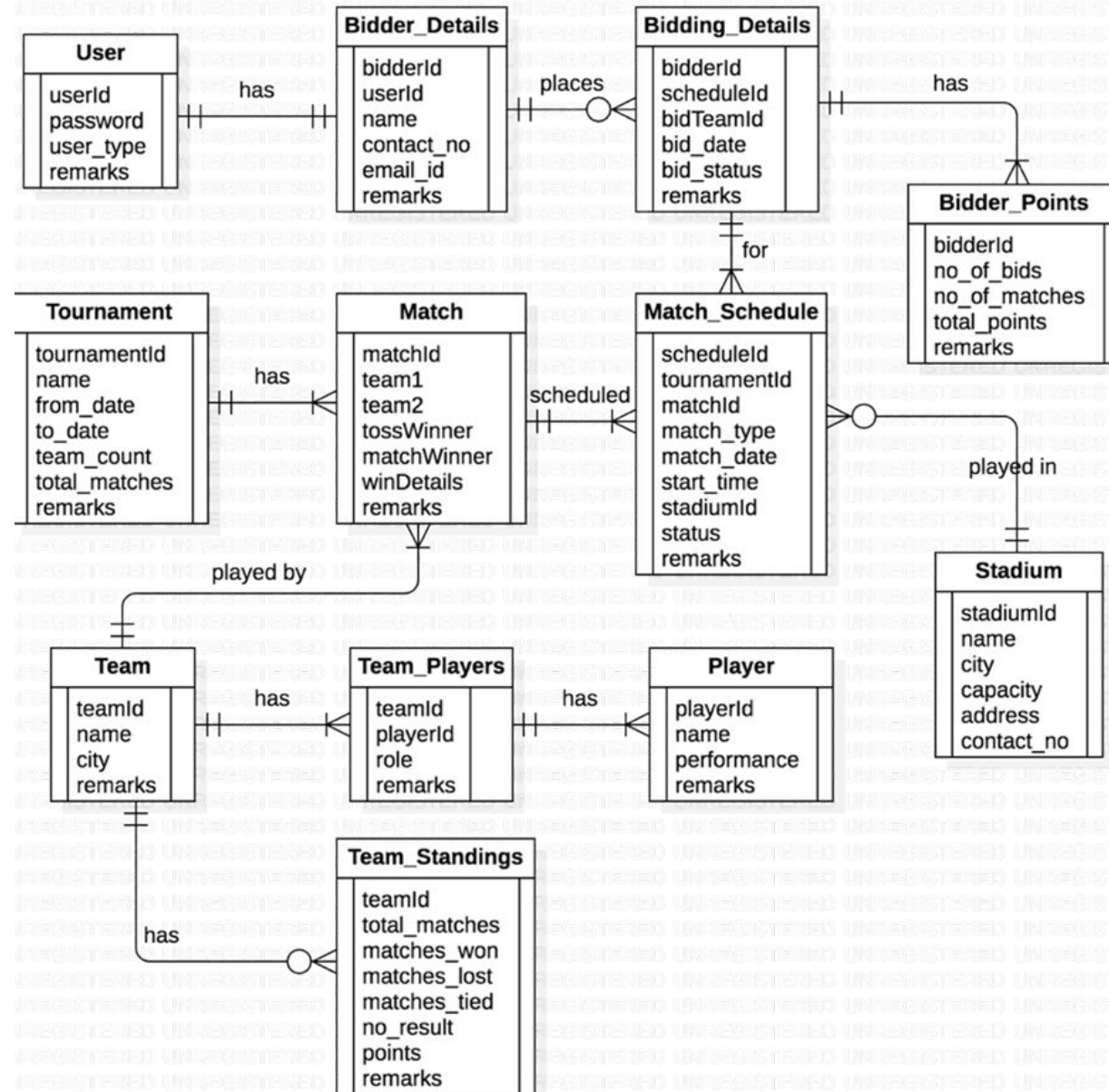
Submitted by :

- Abhilash Jose
- Aishwarya K
- Aishwarya Nandini
- Amit Mishra
- Anush Malipatil

Dataset

Pie-in-the-Sky is a mobile app that is used for bidding for IPL matches legally. Any registered user can bid for any of the IPL matches listed in it. New users or bidders need to register themselves into the app by providing their mobile phone number, email id, and password. Admin will maintain the match roster and keep updating other details in the system.

The app shows the match details which include the playing team, the venue of the match, and the current standing of the teams on the points table. It will display the winner at the end of the match and update the team standings in the tournament and bidder points table. System will send updates to the bidders whenever required. It will also generate the bidders' leaderboard.



1. Show the percentage of wins of each bidder in the order of highest to lowest percentage.

Understanding: The objective is to calculate the total win percentage for each bidder in descending order. To achieve this, a case function is utilized, assigning a value of 1 to 'won' bids. The numerator is then computed based on the 'won' bids, and the denominator is derived by excluding 'cancelled' bids, considering only 'win,' 'lost,' and 'bids.' Subsequently, a 'GROUP BY' operation is performed to aggregate the results

```
select BIDDER_ID,  
count(case when bid_status = "won" then 1 end)as Win,  
count(case when bid_status <> "cancelled" then 1 end) as Total_bids ,  
(count(case when bid_status = "won" then 1 end)/count(case when bid_status <> "cancelled" then 1 end))*100 as Percentage_Win  
from ipl_bidding_details group by BIDDER_ID order by Percentage_Win DESC;
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
BIDDER_ID	Win	Total_bids	Percentage_Win
103	5	5	100.0000
104	5	5	100.0000
121	10	11	90.9091
118	5	6	83.3333
126	4	5	80.0000
117	2	3	66.6667
122	2	3	66.6667

2. Display the number of matches conducted at each stadium with the stadium name and city

Understanding: In this task, the goal is to determine the total number of matches per stadium, along with their respective cities. To achieve this, a join clause and a group clause are employed. Additionally, two cancelled matches at IS Bindra Stadium are excluded from the count.

```
select s.Stadium_id,s.Stadium_Name,s.City,count(m.match_id) as Total_Matches
from ipl_stadium s join ipl_match_schedule m
on s.stadium_id = m.stadium_id
where m.status<> "cancelled"
group by s.stadium_id,s.stadium_name,s.city;
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	Stadium_id	Stadium_Name	City	Total_matches
▶	6	Sawai Mansingh Stadium	Jaipur	10
	7	M. Chinnaswamy Stadium	Bengaluru	13
	3	Eden Gardens	Kolkata	13
	1	Wankhede Stadium	Mumbai	18
	2	Feroz Shah Kotla	Delhi	13
	8	Is Bindra Stadium	Mohali	14
	9	Holkar Stadium	Indore	13
	5	MS Chidambaram Stadium	Chennai	12
	10	MCA Stadium	Pune	7
	4	Rajiv Gandhi International Stadium	Hyderabad	7

3. In a given stadium, what is the percentage of wins by a team which has won the toss?

Understanding: The goal is to find the percentage of wins by a team that won the toss in a specific stadium. Using a CASE statement, a value of 1 is assigned when the toss winner matches the match winner, and 0 otherwise. The results are then calculated with GROUP BY and JOIN operations.

```
select Stadium_ID, Stadium_Name, sum(if(toss_winner = match_winner, 1,0)) as Toss_Wins,  
count(match_id) Total,  
sum(if(toss_winner = match_winner, 1,0)) / count(match_id) * 100 as Percentage_Win  
from ipl_match join ipl_match_schedule using (match_id)  
join ipl_stadium using (stadium_id)  
group by STADIUM_ID, Stadium_Name;
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	Stadium_ID	Stadium_Name	Toss_Wins	Total	Percentage_Win
	6	Sawai Mansingh Stadium	7	10	70.0000
	7	M. Chinnaswamy Stadium	5	13	38.4615
	3	Eden Gardens	5	13	38.4615
	1	Wankhede Stadium	11	18	61.1111
	8	Is Bindra Stadium	10	16	62.5000
	2	Feroz Shah Kotla	7	13	53.8462
	9	Holkar Stadium	5	13	38.4615
	5	MS Chidambaram Stadium	4	12	33.3333

4. Show the total bids along with the bid team and team name.

Understanding: The query provides insights into how often the team that wins the toss also wins the match at each stadium, expressed as a percentage. The results include the stadium ID, name, the number of toss wins, total matches played, and the percentage of wins for each stadium.

```
select count(ibd.bidder_id) as count_bids, it.team_name, ibd.bid_team
from ipl_bidding_details ibd join ipl_bidder_points ibp on ibd.bidder_id=ibp.bidder_id
join ipl_team it on ibd.bid_team=it.team_id
group by it.team_name,ibd.bid_team order by ibd.bid_team;
```

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap Cell Content: 			
	count_bids	team_name	bid_team
▶	22	Chennai Super Kings	1
	26	Delhi Daredevils	2
	24	Kings XI Punjab	3
	22	Kolkata Knight Riders	4
	22	Mumbai Indians	5
	27	Rajasthan Royals	6
	25	Royal Challengers Bangalore	7
	32	Sunrisers Hyderabad	8

5. Show the team id who won the match as per the win details.

Understanding: The task involves finding the team ID of the winners. The match_winner column specifies whether team 1 or team 2 won. A CASE statement is utilized, where if the match column value is 1, the team ID is derived from team_id_1; otherwise, it is extracted from team_id_2



```
select Win_details, SUBSTRING_INDEX(WIN_DETAILS, ' ', 2) Win_Team,  
case when MATCH_WINNER = 1 then TEAM_ID1  
when MATCH_WINNER = 2 then TEAM_ID2  
end as Win_Team_ID from ipl_match;
```

Result Grid			
Filter Rows:		Export:	Wrap Cell Content: IA
	Win_details	Win_Team	Win_Team_ID
▶	Team CSK won by 7 Wkts	Team CSK	1
	Team CSK won by 7 Wkts	Team CSK	1
	Team KKR won by 35 Runs	Team KKR	4
	Team CSK won by 7 Wkts	Team CSK	1
	Team RR won by 35 Runs	Team RR	6
	Team RCB won by 35 Runs	Team RCB	7

6. Display total matches played, total matches won and total matches lost by the team along with its team name.

Understanding: The objective is to obtain information on the total matches played, total matches won, and total matches lost. To achieve this, a GROUP BY operation is applied to aggregate the values. Additionally, a JOIN is utilized to fetch the team names from another table.

```
SELECT iplts.TEAM_ID, TEAM_NAME, sum(MATCHES_PLAYED) as Total_Matches_Played,  
sum(matches_won) as Total_Matches_Won, sum(matches_lost) as Total_Matches_Lost  
FROM ipl_team_standings iplts inner join ipl_team iplt on iplts.TEAM_ID=iplt.TEAM_ID  
group by iplts.TEAM_ID ;
```

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap Cell Content: <input type="checkbox"/>					
	TEAM_ID	TEAM_NAME	Total_Matches_Played	Total_Matches_Won	Total_Matches_Lost
▶	1	Chennai Super Kings	28	18	10
	2	Delhi Daredevils	28	11	17
	3	Kings XI Punjab	28	13	15
	4	Kolkata Knight Riders	28	16	12
	5	Mumbai Indians	28	16	12
	6	Rajasthan Royals	28	16	12
	7	Royal Challengers Bangalore	28	9	18
	8	Sunrisers Hyderabad	28	17	10

7. Display the bowlers for the Mumbai Indians team.

Understanding: The query retrieves information about players who are bowlers and are associated with the team "Mumbai Indians". The `LIKE` operator with `%` is used for pattern matching, allowing for partial matches.

```
select t.team_name,p.player_name,tp.player_role
from ipl_team t join ipl_team_players tp using (TEAM_ID)
join ipl_player p using (PLAYER_ID)
where t.team_name like "%Mumbai indians%" and tp.player_role like "%bowler%";
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	team_name	player_name	player_role
▶	Mumbai Indians	Hardik Pandya	Bowler
	Mumbai Indians	Suryakumar Yadav	Bowler
	Mumbai Indians	Jasprit Bumrah	Bowler
	Mumbai Indians	Evin Lewis	Bowler
	Mumbai Indians	Mayank Markande	Bowler
	Mumbai Indians	Rohit Sharma	Bowler
	Mumbai Indians	Ben Cutting	Bowler
	Mumbai Indians	Kieron Pollard	Bowler
	Mumbai Indians	JP Duminy	Bowler

8. How many all-rounders are there in each team, Display the teams with more than 4

Understanding: The task involves identifying teams with more than 4 all-rounders. To accomplish this, a JOIN is employed to retrieve team-specific columns, and a filter using the LIKE operator is applied. The results are then grouped by team name, and the HAVING clause is used to filter teams with an aggregate count exceeding 4 for all-rounders.

```
select t.team_name,count(tp.player_role) as all_rounder_count from ipl_team t
join ipl_team_players tp using (team_id)
join ipl_player p using (player_id)
where player_role like '%all-rounder%' group by team_name having all_rounder_count>4;
```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	team_name	all_rounder_count			
▶	Delhi Daredevils	7			
	Kings XI Punjab	7			
	Kolkata Knight Riders	5			
	Rajasthan Royals	5			
	Sunrisers Hyderabad	6			

9. Write a query to get the total bidders points for each bidding status of those bidders who bid on CSK when it won the match in M. Chinnaswamy Stadium bidding year-wise

Note the total bidders' points in descending order and the year is bidding year. Display columns: bidding status, bid date as year, total bidder's points

Understanding: The objective is to determine the total points for each bidding team, considering two filter conditions, and grouping the results based on the year. Date functions are utilized to extract the month and year from the bid date. Multiple tables are joined to gather the necessary information and produce the desired output.

```
select ibd.bid_status,year(ibd.bid_date) biddingyear,sum(ibp.total_points) totalbidderpoints
from ipl_bidding_details ibd join ipl_bidder_points ibp on ibd.bidder_id = ibp.bidder_id
join ipl_team it on ibd.bid_team = it.team_id join ipl_match im on it.team_id = im.match_winner
join ipl_match_schedule ims on im.match_id = ims.match_id
join ipl_stadium ist on ims.stadium_id = ist.stadium_id
where it.remarks like "%csk%" and ist.stadium_name like "%M. Chinnaswamy Stadium%" and im.win_details LIKE "%csk won%"
group by ibd.bid_status, biddingyear
order by totalbidderpoints desc, biddingyear;
```

Continuation of Q9

Result Grid				Filter Rows:		Export:	Wrap Cell Content:
	bid_status	biddingyear	totalbidderpoints				
▶	Won	2018	314				
	Won	2017	160				
	Bid	2018	24				
	Cancelled	2017	16				
	Lost	2018	14				
	Bid	2017	10				
	Lost	2017	8				

10. Extract the Bowlers and All Rounders those are in the 5 highest number of wickets.

1. use the `performance_dtls` column from `ipl_player` to get the total number of wickets
2. Do not use the `limit` method because it might not give appropriate results when players have the same number of wickets
3. Do not use joins in any cases.
4. Display the following columns `teamn_name`, `player_name`, and `player_role`.

Understanding: The aim is to extract bowlers and all-rounders ranking within the top 5 based on the highest number of wickets. The query utilizes a subquery with `dense_rank()` to assign ranks, followed by filtering for ranks 1 to 5.

```
select Team_name as Team, Player_name as Player, Player_role as Role
from (select ipl_player.PLAYER_ID, PLAYER_NAME,
dense_rank() over(order by cast(trim(both ' ' from substring_index(substring_index(PERFORMANCE_DTLS,'Dot',1),'wkt-',-1))
as signed int) desc ) as WICKET_RANK,
PLAYER_ROLE, Team_name from ipl_player, ipl_team_players, ipl_team
where ipl_player.PLAYER_ID = ipl_team_players.PLAYER_ID
and PLAYER_ROLE in ('bowler', 'all-rounder')
and ipl_team.TEAM_ID = ipl_team_players.TEAM_ID) as Temp
where WICKET_RANK <= 5;
```

Continuation of Q10

Team	Player	Role
Kings XI Punjab	Andrew Tye	All-Rounder
Sunrisers Hyderabad	Siddarth Kaul	Bowler
Sunrisers Hyderabad	Rashid Khan	Bowler
Royal Challengers Bangalore	Umesh Yadav	All-Rounder
Mumbai Indians	Hardik Pandya	Bowler
Mumbai Indians	Jasprit Bumrah	Bowler
Kolkata Knight Riders	Sunil Narine	All-Rounder
Kolkata Knight Riders	Kuldeep Yadav	All-Rounder

11. Show the percentage of toss wins of each bidder and display the results in descending order based on the percentage

Provide understanding: The goal is to compute the toss win percentage for each bidder and arrange the results in descending order. This is achieved by using a join operation to extract relevant data and applying an if condition to calculate toss win counts. The final output showcases toss win percentages in descending order for each bidder.

```
select sum(if(bid_team = if(toss_winner = 1,team_id1,team_id2) ,1,0)) as no_of_wins, bidder_id,
count(bid_team) count,
round((sum(if(bid_team = if(toss_winner = 1,team_id1,team_id2) ,1,0)) / count(bid_team) ) *100,2) percentage
from ipl_match im join ipl_match_schedule ims using (match_id)
join ipl_bidding_details ibd using(schedule_id)
group by bidder_id order by percentage desc;
```

Result Grid					Filter Rows:		Export:	Wrap Cell Content:
	no_of_wins	bidder_id	count	percentage				
▶	8	110	9	88.89				
	5	118	6	83.33				
	4	124	5	80.00				
	4	126	5	80.00				
	6	105	9	66.67				
	4	115	6	66.67				

12. Find the IPL season which has min duration and max duration.

Output columns should be like the below:




Tournment_ID, Tourment_name, Duration column, Duration

Understanding: The objective is to find the IPL season with the minimum and maximum durations. A table is created using a CTE, filtered for rank 1 for the minimum duration, and a subquery is used to find the season with the last rank for the maximum duration.

```
with temp2 as
(select * , rank() over (order by Total_Tournament_Duration ASC) Duration_rank from
(SELECT TOURNMT_ID, TOURNMT_NAME, datediff(To_date,from_date) as Total_Tournament_Duration
from ipl_tournament) temp1)

select * from temp2 where Duration_rank= 1 or duration_rank = (select max(duration_rank) from temp2);
```

Continuation of Q12

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap Cell Content: 				
	TOURNMT_ID	TOURNMT_NAME	Total_Tournament_Duration	Duration_rank
▶	2009	IPL SEASON - 2009	36	1
	2012	IPL SEASON - 2012	53	10
	2013	IPL SEASON - 2013	53	10

13. Write a query to display to calculate the total points month-wise for the 2017 bid year. sort the results based on total points in descending order and month-wise in ascending order.

Note: Display the following columns:

1. Bidder ID, 2. Bidder Name, 3. bid date as Year, 4. bid date as Month, 5. Total points

Only use joins for the above query queries.

Provide understanding: The code retrieves a summary of bidding activities in 2017 and the required data by joining three tables and groups the results by bidder and time, and then orders the output by total points in descending order and bid month in ascending order.

```
select distinct bdr.BIDDER_ID,bdr.BIDDER_NAME,year(bds.BID_DATE) as Year,
month(bds.BID_DATE) as Month,pt.TOTAL_POINTS as Total_Points
from ipl_bidder_details bdr inner join ipl_bidder_points pt on bdr.BIDDER_ID=pt.BIDDER_ID
inner join ipl_bidding_details bds
on pt.BIDDER_ID=bds.BIDDER_ID
where year(bds.BID_DATE)=2017
order by Total_Points desc,Month asc ;
```

Continuation of Q13

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap Cell Content: 					
	BIDDER_ID	BIDDER_NAME	Year	Month	Total_Points
▶	121	Aryabhatta Parachure	2017	4	35
	121	Aryabhatta Parachure	2017	5	35
	103	Megaduta Dheer	2017	4	19
	103	Megaduta Dheer	2017	5	19
	104	Chatur Mahalanabis	2017	4	17
	104	Chatur Mahalanabis	2017	5	17
	118	Akshara Pandey	2017	4	15
	110	Mishri Nayar	2017	4	15
	110	Akshara Pandey	2017	5	15

14. Write a query for the above question using sub queries by having the same constraints as the above question.

Note: Display the following columns:

1. Bidder ID, 2. Bidder Name, 3. bid date as Year, 4. bid date as Month, 5. Total points

Provide understanding: The code accomplishes the goal of summarizing bidding activities in 2017, using subquery in the SELECT clause to calculate the total points for each bidder. Filters results for the year 2017, groups by bidder and time, and orders the output by total points in descending order and bid month in ascending order.

```
select bidder_id, (select bidder_name from ipl_bidder_details
where ipl_bidder_details.bidder_id=ipl_bidding_details.bidder_id) as bidder_name,
year(bid_date) as `year`, monthname(bid_date) as `month`,
(select total_points from ipl_bidder_points
where ipl_bidder_points.bidder_id=ipl_bidding_details.bidder_id) as total_points from ipl_bidding_details
where year(bid_date)=2017
group by bidder_id,bidder_name,year,month,total_points
order by total_points desc, Month asc ;
```

Continuation of Q14

	bidder_id	bidder_name	year	month	total_points
▶	121	Aryabhatta Parachure	2017	April	35
	121	Aryabhatta Parachure	2017	May	35
	103	Megaduta Dheer	2017	April	19
	103	Megaduta Dheer	2017	May	19
	104	Chatur Mahalanabis	2017	April	17
	104	Chatur Mahalanabis	2017	May	17
	118	Akshara Pandey	2017	April	15
	110	Mishri Nayar	2017	April	15
	110	Akshara Pandey	2017	May	15

15. Write a query to get the top 3 and bottom 3 bidders based on the total bidding points for the 2018 bidding year.

Output columns should be like:

Bidder Id, Ranks (optional), Total points, Highest_3_Bidders --> columns contains name of bidder, Lowest_3_Bidders --> columns contains name of bidder;

Understanding: In this scenario, the objective is to identify the top 3 and bottom 3 bidders. To achieve this, a temporary table is created using a Common Table Expression (CTE) along with window functions such as 'RANK.' Two derived tables are then formed—one to determine the top 3 and another to identify the bottom 3. These two tables are combined using the 'UNION' operation to create a unified result set.

```
with temp as(
  SELECT *,
  rank() over( order by TOTAL_POINTS Desc) Points_Rank,
  (select bidder_name from ipl_bidder_details ipl_bd where ipl_bd.bidder_id = ipl_bp.bidder_id) as Bidder_Name
  FROM ipl.ipl_bidder_points ipl_bp where TOURNMT_ID=2018)

  (select Bidder_id, points_rank, total_points, bidder_name from temp order by points_rank ASC limit 3)
  union
  (select Bidder_id, points_rank, total_points, bidder_name from temp order by points_rank DESC limit 3);
```


Continuation of Q15

	Bidder_id	points_rank	total_points	bidder_name
▶	121	1	35	Aryabhatta Parachure
	103	2	19	Megaduta Dheer
	104	3	17	Chatur Mahalanabis
	102	28	0	Krishan Valimbe
	109	28	0	Gagan Panda
	116	28	0	Ronald D'Souza

16. Create two tables called Student_details and Student_details_backup.

Output columns should be: like:

Bidder Id, Ranks (optional), Total points, Highest_3_Bidders --> columns contains name of bidder, Lowest_3_Bidders --> columns contains name of bidder;

Assume you are working in an Ed-tech company namely Great Learning where you will be inserting and modifying the details of the students in the Student details table. Every time the students changed their details like mobile number, You need to update their details in the student details table. Here is one thing you should ensure whenever the new students' details come , you should also store them in the Student backup table so that if you modify the details in the student details table, you will be having the old details safely.

You need not insert the records separately into both tables rather Create a trigger in such a way that It should insert the details into the Student back table when you inserted the student details into the student table automatically.

Understanding: The task is to create two tables and enable triggers to update them automatically. One table is the 'Student' table, and the other is a backup table for the 'Student' table. Two triggers are created: one to update the backup table as soon as we update the 'Student' table, and another to insert new values for updates in the backup table. Both triggers work together to ensure that the backup table is updated automatically

Continuation of Q16

-- Step 1

-- Creating Database and also Tables

Create database GreatLearning;

Use Greatlearning;

```
CREATE TABLE Student_details (  
    Student_id INT PRIMARY KEY,  
    Student_name VARCHAR(250),  
    Mail_id VARCHAR(200),  
    Mobile_no VARCHAR(20)  
);
```

-- Create the Student_details_backup table

```
CREATE TABLE Student_details_backup (  
    Student_id INT PRIMARY KEY,  
    Student_name VARCHAR(250),  
    Mail_id VARCHAR(200),  
    Mobile_no VARCHAR(20)  
);
```

-- Step 2

DROP TRIGGER IF EXISTS after_student_update; #optional

-- Just to make sure that no triggers are created before

We are Creating 2 Triggers one for insert and another for Update

-- Trigger for Insert

DELIMITER //

CREATE TRIGGER insert_student_details_trigger

AFTER INSERT ON Student_details

FOR EACH ROW

BEGIN

INSERT INTO Student_details_backup (student_id, student_name, mail_id, mobil

VALUES (NEW.student_id, NEW.student_name, NEW.mail_id, NEW.mobile_no)

ON DUPLICATE KEY UPDATE

student_name = NEW.student_name,

mail_id = NEW.mail_id,

mobile_no = NEW.mobile_no;

END;

//

DELIMITER ;

Continuation of Q16

```
-- Trigger for Update
```

```
DELIMITER //
```

```
CREATE TRIGGER update_student_details_trigger  
AFTER UPDATE ON Student_details  
FOR EACH ROW
```

```
BEGIN
```

```
    UPDATE Student_details_backup  
    SET  
        student_name = NEW.student_name,  
        mail_id = NEW.mail_id,  
        mobile_no = NEW.mobile_no  
    WHERE student_id = NEW.student_id;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
-- Step 3
```

```
-- Checking if values change upon updating or inserting
```

```
-- Inserting values
```

```
INSERT INTO Student_details (Student_id, Student_name, Mail_id, Mobile_no)  
VALUES
```

```
(1, 'Amit Mishra', 'amit.mishra@yahoo.com', '8991234567'),  
(2, 'Anush Malipatil', 'anush.malipatil@yahoo.com', '9882345678'),  
(3, 'Aishwarya N', 'aishwarya_n@outlook.com', '9993456789'),  
(4, 'Aishwarya K', 'aishwarya_k@outlook.com', '7894567890'),  
(5, 'Abhilash Jose', 'abhilash.jose@yahoo.com', '6905678901');
```

Continuation of Q16

```
-- Step 4
-- updating Aishwarya K's number to check if the backup is also updating

UPDATE Student_details
SET Mobile_no = '9022221111'
WHERE Student_name = 'Aishwarya K';

-- Step 5
-- insert a new student to check to student_details (This create automatic update in backup table)
INSERT INTO Student_details
VALUES (6, 'Amitha K', 'amitha.k@yahoo.com', '8990000561');
```


Conclusions

- Navigated through crucial database concepts, including Joins, Sub-Queries, Window functions, CTE, and Triggers, fostering a practical understanding of their applications.
- By leveraging these concepts, we have the capability to craft queries that generate the necessary datasets, enabling us to play a pivotal role in shaping critical decisions

Thank You !!!....