FINAL PROJECT

1. **Create a table named ‘matches’ with appropriate data types for columns**

```
CREATE TABLE matches

(id int , city varchar , date date , player_of_match varchar , venue varchar ,

neutral_venue int , team1 varchar , team2 varchar , toss_winner varchar ,
toss_decision varchar ,

winner varchar , result varchar , result_margin int , eliminator varchar ,
method varchar , umpire1 varchar , umpire2 varchar);
```

```
CREATE TABLE Query returned successfully in 91 msec.
```

2. **Create a table named ‘deliveries’ with appropriate data types for columns**

```
CREATE TABLE deliveries

(id int , inning int , over int , ball int , batsman varchar , non_striker varchar ,
bowler varchar ,

 batsman_runs int , extra_runs int , total_runs int , is_wicket int ,
dismissal_kind varchar ,

 player_dismissed varchar , fielder varchar , extras_type varchar ,
batting_team varchar , bowling_team varchar);
```

```
CREATE TABLE Query returned successfully in 91 msec.
```

3. **Import data from csv file ’IPL_matches.csv’ attached in resources to the table ‘matches’ which was created in Q1**

```
copy matches from 'C:\Program
Files\PostgreSQL\14\data\Data\IPLMatches+IPLBall\IPL_matches.csv'
delimiter ',' csv header;
```

```
COPY 816 Query returned successfully in 121 msec.
```

4. **Import data from csv file ’IPL_Ball.csv’ attached in resources to the table ‘deliveries’ which was created in Q2**

```
copy deliveries from 'C:\Program
Files\PostgreSQL\14\data\Data\IPLMatches+IPLBall\IPL_Ball.csv' delimiter ','
csv header;
```

```
COPY 193468 Query returned successfully in 1 secs 674 msec.
```

5. **Select the top 20 rows of the *deliveries* table after ordering them by id, inning, over, ball in ascending order.**

```
select * from deliveries order by id,inning,over,ball limit 20;
```

```
Successfully run. Total query runtime: 274 msec. 20 rows affected.
```

6. **Select the top 20 rows of the *matches* table.**

```
select * from matches limit 20;
```

```
Successfully run. Total query runtime: 152 msec. 20 rows affected.
```

7. **Fetch data of all the matches played on 2nd May 2013 from the *matches* table..**

```
select * from matches where date = '2013-05-02';
```

```
Successfully run. Total query runtime: 152 msec. 2 rows affected.
```

8. **Fetch data of all the matches where the result mode is ‘runs’ and margin of victory is more than 100 runs.**

```
select * from matches where result = 'runs' and result_margin > 100;
```

```
Successfully run. Total query runtime: 111 msec. 9 rows affected.
```

9. **Fetch data of all the matches where the final scores of both teams tied and order it in descending order of the date.**

```
select * from matches where result = 'tie' order by date desc;
```

```
Successfully run. Total query runtime: 142 msec. 13 rows affected.
```

10. **Get the count of cities that have hosted an IPL match.**

```
select count(distinct city) as city_count from matches;
```

(33)

```
Successfully run. Total query runtime: 96 msec. 1 rows affected.
```

11. **Create table *deliveries_v02* with all the columns of the table '*deliveries'* and an additional column *ball_result* containing values *boundary*, *dot* or *other* depending on the *total_run* (boundary for >= 4, dot for 0 and other for any other number)**
(Hint 1 : CASE WHEN statement is used to get condition based results)
(Hint 2: To convert the output data of select statement into a table, you can use a subquery. Create table *table_name* as *[entire select statement].*

```
create table deliveries_v02 as select * ,

case when total_runs >=4 then 'boundary'

when total_runs = 0 then 'dot'

else 'other' end as ball_result from deliveries;
```

```
SELECT 193468 Query returned successfully in 1 secs 508 msec.
```

12. **Write a query to fetch the total number of boundaries and dot balls from the *deliveries_v02* table.**

```
select ball_result, count(*) from deliveries_v02 where ball_result in
('boundary','dot') group by ball_result;
```

boundary = 31468 || dot = 67841

```
Successfully run. Total query runtime: 247 msec. 2 rows affected.
```

13. **Write a query to fetch the total number of boundaries scored by each team from the *deliveries_v02* table and order it in descending order of the number of boundaries scored.**

```
select batting_team as team,count(*) as boundary_count from deliveries_v02
where ball_result = 'boundary' group by team order by boundary_count desc;
```

```
Successfully run. Total query runtime: 223 msec. 15 rows affected.
```

14. **Write a query to fetch the total number of dot balls bowled by each team and order it in descending order of the total number of dot balls bowled.**

```
select bowling_team as team,count(*) as dot_count from deliveries_v02
where ball_result = 'dot' group by team order by dot_count desc;
```

```
Successfully run. Total query runtime: 207 msec. 16 rows affected.
```

15. **Write a query to fetch the total number of dismissals by dismissal kinds where dismissal kind is not NA**

```
select dismissal_kind, count(*) as total_no_of_dismissal from deliveries where
dismissal_kind not like 'NA' group by dismissal_kind;
```

```
Successfully run. Total query runtime: 241 msec. 9 rows affected.
```

16. **Write a query to get the top 5 bowlers who conceded maximum extra runs from the *deliveries* table**

```
select bowler, sum(extra_runs) as total_extra_runs from deliveries group by
bowler order by total_extra_runs desc limit 5;
```

```
Successfully run. Total query runtime: 196 msec. 5 rows affected.
```

17. **Write a query to create a table named *deliveries_v03* with all the columns of *deliveries_v02* table and two additional column (named *venue* and *match_date*) of *venue* and *date* from table *matches***

```
create table deliveries_v03 as select d.*,m.venue,m.date as match_date from
deliveries_v02 as d left join matches as m on d.id = m.id;
```

```
SELECT 193468 Query returned successfully in 1 secs 736 msec.
```

18. **Write a query to fetch the total runs scored for each venue and order it in the descending order of total runs scored.**

```
select venue, sum(total_runs) as total_runs_scored from deliveries_v03 group
by venue order by total_runs_scored desc;
```

```
Successfully run. Total query runtime: 208 msec. 36 rows affected.
```

19. **Write a query to fetch the year-wise total runs scored at *Eden Gardens* and order it in the descending order of total runs scored.**

```
select extract(year from match_date) as year , sum(total_runs) as
total_runs_scored from deliveries_v03 where venue = 'Eden Gardens' group
by year order by total_runs_scored desc;

Successfully run. Total query runtime: 99 msec. 11 rows affected.
```

20. **Get unique team1 names from the *matches* table, you will notice that there are two entries for *Rising Pune Supergiant* one with *Rising Pune Supergiant* and another one with *Rising Pune Supergiants*. Your task is to create a *matches_corrected* table with two additional columns *team1_corr* and *team2_corr* containing team names with replacing *Rising Pune Supergiants* with *Rising Pune Supergiant*. Now analyse these newly created columns.**

```
create table matches_corrected as select * ,

case when team1 = 'Rising Pune Supergiants' then 'Rising Pune Supergiant'
else team1 end as team1_corr ,

case when team2 = 'Rising Pune Supergiants' then 'Rising Pune Supergiant'
else team2 end as team2_corr from matches;

SELECT 816 Query returned successfully in 60 msec.
```

21. **Create a new table deliveries_v04 with the first column as ball_id containing information of match_id, inning, over and ball separated by '-' (For ex. 335982-1-0-1 match_id-inning-over-ball) and rest of the columns same as deliveries_v03)**

```
create table deliveries_v04 as select id||'-'||inning||'-'||over||'-'||ball as ball_id , *
from deliveries_v03;

SELECT 193468 Query returned successfully in 825 msec.
```

22. **Compare the total count of rows and total count of distinct ball_id in deliveries_v04;**

```
select count(*) as total_row_count , count(distinct ball_id) as
total_distinct_ball_id_count from deliveries_v04;

total_row_count : 193468 || total_distinct_ball_id_count : 193458

Successfully run. Total query runtime: 885 msec. 1 rows affected.
```

23. **SQL Row_Number() function is used to sort and assign row numbers to data rows in the presence of multiple groups. For example, to identify the top 10 rows which have the highest order amount in each region, we can use row_number to assign row numbers in each group (region) with any particular order (decreasing order of order amount) and then we can use this new column to apply filters. Using this knowledge, solve the following exercise. You can use hints to create an additional column of row number.**
Create table deliveries_v05 with all columns of deliveries_v04 and an additional column for row number partition over ball_id. (HINT : Syntax to add along with other columns, row_number() over (partition by ball_id) as r_num)

```
create table deliveries_v05 as select *, row_number() over (partition by ball_id order by ball_id) as r_num from deliveries_v04;
```

```
SELECT 193468 Query returned successfully in 3 secs 854 msec.
```

24. **Use the r_num created in deliveries_v05 to identify instances where ball_id is repeating. (HINT : select * from deliveries_v05 WHERE r_num=2;)**

```
select * from deliveries_v05 where r_num > 1;
```

```
Successfully run. Total query runtime: 348 msec. 10 rows affected.
```

25. **Use subqueries to fetch data of all the ball_id which are repeating. (HINT: SELECT * FROM deliveries_v05 WHERE ball_id in (select BALL_ID from deliveries_v05 WHERE r_num=2);**

```
select * from deliveries_v05 where ball_id in (select ball_id from deliveries_v05 where r_num > 1);
```

```
Successfully run. Total query runtime: 218 msec. 20 rows affected.
```