DBMS - Assignment - 3 Report

Team Details:

K K Tarun Kumar– PES1UG19CS222
Koushik Varma Mandapati – PES1UG19CS230
Kosaraju Bhargava Krishna – PES1UG19CS225

Project Title: Event Registration Management System

Simple Queries:

1) Display the Marketer who has the ID = MAR001.

```
postgres=# \c event_management_sys
You are now connected to database "event_management_sys" as user "postgres".
event_management_sys=# select * from Marketer WHERE Marketer_ID = 'MAR001';
 marketer_id | username | password_ | referal_code
-------------+----------+-----------+--------------
 MAR001      | John     | john123   | REF001
(1 row)
```

2) Display the set of Events which has an event price greater than 500.

```
event_management_sys=# select * from Event WHERE Event_Price > 500;
   street    |  city  | pincode | event_id | start_date |  end_date  | event_description | event_price | event_capacity |    event_category    | organizer_id
-------------+--------+---------+----------+------------+------------+-------------------+-------------+----------------+----------------------+--------------
 Bhandra West | Mumbai |  560007 | EID003   | 2021-12-11 | 2021-12-13 | Smooth Play       |         650 |             90 | Table Tennis         | ORG001
 Gandhinagar  | Delhi  |  560069 | EID004   | 2021-11-15 | 2021-11-19 | Hamilton          |         600 |            100 | Open Theater Musical | ORG002
(2 rows)
```

3) Display the Participants who are Female.

```
event_management_sys=# select * from Participant_ WHERE Sex = 'F';
 participant_id | age | sex | f_name |  l_name  |         mail_id         |  username   | password_ | marketer_id | team_id
----------------+-----+-----+--------+----------+-------------------------+-------------+-----------+-------------+---------
 PRT002         |  25 | F   | Deepa  | Girish   | deepagirish@avdva.com   | deepa86     | deepa123  | MAR002      |
 PRT003         |  33 | F   | Nidhi  | Burman   | nidhiburman@ersfg.com   | nidhirokz34 | nidhi123  | MAR003      | TEM002
 PRT006         |  29 | F   | Pooja  | Pattekar | poojapattekar@erchg.com | poojamallu  | pooja123  | MAR003      | TEM003
(3 rows)
```

4) Display the set of Payment methods which are either done through UPI or Card.

```
event_management_sys=# select * from Payment_Details WHERE (Payment_Method = 'UPI' or Payment_Method = 'Card');
 payment_id | payment_method | participant_id
------------+----------------+----------------
 PYM001     | Card           | PRT001
 PYM002     | UPI            | PRT002
 PYM004     | UPI            | PRT004
 PYM005     | Card           | PRT005
(4 rows)
```

5) Display the set of Tickets which has a class tier of S.

```
event_management_sys=# select * from Ticket WHERE class_tier = 'S';
 ticket_no | seat_no | class_tier | payment_id
-----------+---------+------------+------------
 TIC001    | SET001  | S          | PYM001
 TIC005    | SET005  | S          | PYM005
 TIC006    | SET006  | S          | PYM006
(3 rows)
```

6) Display the set of teams who have team size greater than 1.

```
event_management_sys=# select * from Team WHERE Team_Size > 1;
 team_id |   team_name    | team_size
---------+----------------+-----------
 TEM001  | The Insiders |         2
(1 row)
```

Complex Queries:

1) Select all participants and display their full name with mail id and team id and sort by the payment ID.

```
event_management_sys=# SELECT participant_.participant_id, f_name, l_name, mail_id, team_id,payment_id FROM participant_ INNER JOIN payment_details ON payment_details.participant_id = participant_.participant_id
 ORDER BY payment_id;
 participant_id |  f_name  |  l_name  |        mail_id         | team_id | payment_id
----------------+----------+----------+------------------------+---------+------------
 PRT001         | Rishab   | Kumar    | rishabkumar@kjsdv.com  |         | PYM001
 PRT002         | Deepa    | Girish   | deepagirish@avdva.com  |         | PYM002
 PRT003         | Nidhi    | Burman   | nidhiburman@ersfg.com  | TEM002  | PYM003
 PRT004         | Kaushal  | Reddy    | kaushalreddy@ersfg.com | TEM001  | PYM004
 PRT005         | Janardhan | Chethri | janardhancheth@ekffg.com | TEM001 | PYM005
 PRT006         | Pooja    | Pattekar | poojapattekar@erchg.com | TEM003 | PYM006
(6 rows)
```

2) Select all Participants who are above the age of 25.

```
event_management_sys=# SELECT * FROM participant_ except ( SELECT * FROM participant_ WHERE age <= '25' );
 participant_id | age | sex | f_name |  l_name  |         mail_id         |  username   | password_ | marketer_id | team_id
----------------+-----+-----+--------+----------+-------------------------+-------------+-----------+-------------+---------
 PRT006         |  29 | F   | Pooja  | Pattekar | poojapattekar@erchg.com | poojamallu  | pooja123  | MAR003      | TEM003
 PRT003         |  33 | F   | Nidhi  | Burman   | nidhiburman@ersfg.com   | nidhirokz34 | nidhi123  | MAR003      | TEM002
(2 rows)
```

Approach (with performance analysis and query execution plans):

This problem can be done using two different queries as shown below along with different query plans associated at each stage and cost of the query operation. In the first screenshot we have used set operation except clause. In the second screenshot we have used not in clause. We can see that the last query execution plans are different.

For the except clause, last query execution plan is HashsetOp + Append + Subquery Scan and for the not in clause, it's a Sequential Scan. In terms of time, sequential scan due to its iterative nature takes longer and hence has a higher cost. Hence the query using set operation except clause is the optimal solution for the given problem.

```
event_management_sys=# explain SELECT * FROM participant_ except ( SELECT * FROM participant_ WHERE age <= '25' );
                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
 HashSetOp Except  (cost=0.00..35.18 rows=200 width=380)
   ->  Append  (cost=0.00..28.51 rows=267 width=380)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..14.00 rows=200 width=380)
               ->  Seq Scan on participant_  (cost=0.00..12.00 rows=200 width=376)
         ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..13.17 rows=67 width=380)
               ->  Seq Scan on participant_ participant__1  (cost=0.00..12.50 rows=67 width=376)
                     Filter: (age <= 25)
(7 rows)


event_management_sys=# explain SELECT * FROM participant_ WHERE age not IN ( SELECT age FROM participant_ WHERE age <= '25' );
                                QUERY PLAN
---------------------------------------------------------------------------
 Seq Scan on participant_  (cost=12.67..25.17 rows=100 width=376)
   Filter: (NOT (hashed SubPlan 1))
   SubPlan 1
     ->  Seq Scan on participant_ participant__1  (cost=0.00..12.50 rows=67 width=4)
           Filter: (age <= 25)
(5 rows)
```

3) Find out the name of the participant the ticket belongs to with just ticket_no (TIC003).

```
event_management_sys=# select participant_.f_name, participant_.l_name,t.ticket_no from participant_ INNER JOIN payment_
details on participant_.participant_ID = payment_details.participant_id INNER JOIN TICKET t ON Payment_details.Payment_i
d = t.payment_id WHERE t.ticket_no = 'TIC003';
 f_name | l_name | ticket_no
--------+--------+-----------
 Nidhi  | Burman | TIC003
(1 row)
```

4) Find out all the participants that belong to teams.

```
event_management_sys=# select * from participant_ where team_id in (select team_id from Participant_ intersect select team.team_id from team);
 participant_id | age | sex |  f_name   |  l_name  |         mail_id          |    username     | password_  | marketer_id | team_id
----------------+-----+-----+-----------+----------+--------------------------+-----------------+------------+-------------+---------
 PRT003         |  33 | F   | Nidhi     | Burman   | nidhiburman@ersfg.com    | nidhirokz34     | nidhi123   | MAR003      | TEM002
 PRT004         |  22 | M   | Kaushal   | Reddy    | kaushalreddy@ersfg.com   | kaushalbling    | kaushal123 | MAR002      | TEM001
 PRT005         |  21 | M   | Janardhan | Chethri  | janardhancheth@ekffg.com | janardhanbling2 | jannu123   | MAR002      | TEM001
 PRT006         |  29 | F   | Pooja     | Pattekar | poojapattekar@erchg.com  | poojamallu      | pooja123   | MAR003      | TEM003
(4 rows)
```

5) Find out all the participants going to events in Bangalore.

```
event_management_sys=# select p.f_name, p.l_name,e.event_description,e.start_date, e.city from participant_ p INNER JOIN
 Payment_details pd ON p.participant_id = Pd.participant_id INNER JOIN for_an_ f ON pd.Payment_ID = f.Payment_ID INNER J
OIN event e ON e.event_id = f.event_id where e.city='Bangalore';
  f_name   | l_name  |                 event_description                 | start_date |   city
-----------+---------+---------------------------------------------------+------------+-----------
 Rishab    | Kumar   | Most Interesting Person in the Room-Kenny Sebastian | 2021-12-02 | Bangalore
 Janardhan | Chethri | Most Interesting Person in the Room-Kenny Sebastian | 2021-12-02 | Bangalore
(2 rows)
```

6) Find the number of events currently present by their city (subtotal) and order by all city and category(total) and price to attend all events in a city.

```
event_management_sys=# select city, event_category, event_description, sum(event_price) from event e group by ROLLUP(e.city,e.event_category,event_description) ORDER BY city, event_category;
   city    |    event_category    |                 event_description                 | sum
-----------+----------------------+---------------------------------------------------+------
 Bangalore | Comedy               | Most Interesting Person in the Room-Kenny Sebastian |  450
 Bangalore | Comedy               |                                                   |  450
 Bangalore | Hopscotch            | Game of Hops                                      |   50
 Bangalore | Hopscotch            |                                                   |   50
 Bangalore | Live Concert         | Benny Dayal Live                                  |  250
 Bangalore | Live Concert         |                                                   |  250
 Bangalore |                      |                                                   |  750
 Delhi     | Open Theater Musical | Hamilton                                          |  600
 Delhi     | Open Theater Musical |                                                   |  600
 Delhi     |                      |                                                   |  600
 Kolkata   | Musical/Comedy       | Inside - Bo Burnham                               |  500
 Kolkata   | Musical/Comedy       |                                                   |  500
 Kolkata   |                      |                                                   |  500
 Mumbai    | Table Tennis         | Smooth Play                                       |  650
 Mumbai    | Table Tennis         |                                                   |  650
 Mumbai    |                      |                                                   |  650
           |                      |                                                   | 2500
(17 rows)
```

Database Access Privilege Levels:

ACCESS PRIVILEGES:

Creating 3 users (Organizer, Marketer, Participant), each of them having different access privilege levels in the database based on their roles in an Event and Displaying their respective privileges.

ORGANIZER:

```
event_management_sys=# create user organizer1 with password 'o1234';
CREATE ROLE
event_management_sys=# grant all on Participant_, Marketer, Organizer, Event, Team, Ticket, Register, Payment_Details to organizer1;
GRANT
```

```
event_management_sys=# select * from information_schema.role_table_grants WHERE grantee='organizer1';
 grantor  |  grantee   |    table_catalog     | table_schema |   table_name    | privilege_type | is_grantable | with_hierarchy
----------+------------+----------------------+--------------+-----------------+----------------+--------------+----------------
 postgres | organizer1 | event_management_sys | public       | participant_    | INSERT         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | participant_    | SELECT         | NO           | YES
 postgres | organizer1 | event_management_sys | public       | participant_    | UPDATE         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | participant_    | DELETE         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | participant_    | TRUNCATE       | NO           | NO
 postgres | organizer1 | event_management_sys | public       | participant_    | REFERENCES     | NO           | NO
 postgres | organizer1 | event_management_sys | public       | participant_    | TRIGGER        | NO           | NO
 postgres | organizer1 | event_management_sys | public       | marketer        | INSERT         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | marketer        | SELECT         | NO           | YES
 postgres | organizer1 | event_management_sys | public       | marketer        | UPDATE         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | marketer        | DELETE         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | marketer        | TRUNCATE       | NO           | NO
 postgres | organizer1 | event_management_sys | public       | marketer        | REFERENCES     | NO           | NO
 postgres | organizer1 | event_management_sys | public       | marketer        | TRIGGER        | NO           | NO
 postgres | organizer1 | event_management_sys | public       | organizer       | INSERT         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | organizer       | SELECT         | NO           | YES
 postgres | organizer1 | event_management_sys | public       | organizer       | UPDATE         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | organizer       | DELETE         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | organizer       | TRUNCATE       | NO           | NO
 postgres | organizer1 | event_management_sys | public       | organizer       | REFERENCES     | NO           | NO
 postgres | organizer1 | event_management_sys | public       | organizer       | TRIGGER        | NO           | NO
 postgres | organizer1 | event_management_sys | public       | event           | INSERT         | NO           | NO
 postgres | organizer1 | event_management_sys | public       | event           | SELECT         | NO           | YES
       | NO
(56 rows)
```

MARKETER:

```
event_management_sys=# create user marketer1 with password 'mkt#1234';
CREATE ROLE
event_management_sys=# grant select on Event,Participant_ to marketer1;
GRANT
event_management_sys=# grant insert,delete,update on Marketer to marketer1;
GRANT
event_management_sys=# select * from information_schema.role_table_grants WHERE grantee='marketer1';
 grantor  |  grantee   |    table_catalog     | table_schema |  table_name  | privilege_type | is_grantable | with_hierarchy
----------+------------+----------------------+--------------+--------------+----------------+--------------+----------------
 postgres | marketer1 | event_management_sys | public       | marketer     | INSERT         | NO           | NO
 postgres | marketer1 | event_management_sys | public       | marketer     | UPDATE         | NO           | NO
 postgres | marketer1 | event_management_sys | public       | marketer     | DELETE         | NO           | NO
 postgres | marketer1 | event_management_sys | public       | event        | SELECT         | NO           | YES
 postgres | marketer1 | event_management_sys | public       | participant_ | SELECT         | NO           | YES
(5 rows)
```

PARTICIPANT:

```
event_management_sys=# create user participant1 with password 'prt#1234';
CREATE ROLE
event_management_sys=# grant select on Ticket, Event to participant1;
GRANT
event_management_sys=# grant insert,update on Team, Payment_Details, Register to participant1;
GRANT
event_management_sys=# select * from information_schema.role_table_grants WHERE grantee='participant1';
 grantor  |  grantee    |    table_catalog     | table_schema |   table_name    | privilege_type | is_grantable | with_hierarchy
----------+-------------+----------------------+--------------+-----------------+----------------+--------------+----------------
 postgres | participant1 | event_management_sys | public       | ticket          | SELECT         | NO           | YES
 postgres | participant1 | event_management_sys | public       | event           | SELECT         | NO           | YES
 postgres | participant1 | event_management_sys | public       | team            | INSERT         | NO           | NO
 postgres | participant1 | event_management_sys | public       | team            | UPDATE         | NO           | NO
 postgres | participant1 | event_management_sys | public       | payment_details | INSERT         | NO           | NO
 postgres | participant1 | event_management_sys | public       | payment_details | UPDATE         | NO           | NO
 postgres | participant1 | event_management_sys | public       | register        | INSERT         | NO           | NO
 postgres | participant1 | event_management_sys | public       | register        | UPDATE         | NO           | NO
(8 rows)
```

• Connecting to the database as each user to illustrate privilege levels:

User – Organizer:

- Has all the privileges such as view, modify and delete

```
postgres=# \c event_management_sys organizer1
Password for user organizer1:
You are now connected to database "event_management_sys" as user "organizer1".
event_management_sys=> select * from Event;
      street      |   city    | pincode | event_id | start_date |  end_date  |                event_description                | event_price | event_capacity |    event_category    | organizer_id
------------------+-----------+---------+----------+------------+------------+-------------------------------------------------+-------------+----------------+----------------------+--------------
 MG road          | Bangalore |  560003 | EID001   | 2021-12-02 | 2021-12-03 | Most Interesting Person in the Room-Kenny Sebastian |         450 |             70 | Comedy               | ORG001
 Gulab Jamun Road | Kolkata   |  560098 | EID002   | 2021-11-09 | 2021-11-10 | Inside - Bo Burnham                             |         500 |             90 | Musical/Comedy       | ORG001
 Bhandra West     | Mumbai    |  560007 | EID003   | 2021-12-11 | 2021-12-13 | Smooth Play                                     |         650 |             90 | Table Tennis         | ORG001
 Gandhinagar      | Delhi     |  560069 | EID004   | 2021-11-15 | 2021-11-19 | Hamilton                                        |         600 |            100 | Open Theater Musical | ORG002
 100 ft Ring Road | Bangalore |  560010 | EID005   | 2022-01-16 | 2021-01-16 | Benny Dayal Live                                |         250 |            500 | Live Concert         | ORG002
 150 ft Ring Road | Bangalore |  560047 | EID006   | 2022-02-20 | 2021-02-22 | Game of Hops                                    |          50 |             30 | Hopscotch            | ORG003
(6 rows)


event_management_sys=> insert into Team(Team_ID,Team_Name,Team_Size) values ('TEM004','Slytherin',4);
INSERT 0 1
event_management_sys=> select * from Team;
 team_id |  team_name   | team_size
---------+--------------+-----------
 TEM001  | The Insiders |         2
 TEM002  | Loner        |         1
 TEM003  | FC Barcelona |         1
 TEM004  | Slytherin    |         4
(4 rows)


event_management_sys=> delete from Team where Team_Size = 4;
DELETE 1
event_management_sys=> select * from Team;
 team_id |  team_name   | team_size
---------+--------------+-----------
 TEM001  | The Insiders |         2
 TEM002  | Loner        |         1
 TEM003  | FC Barcelona |         1
(3 rows)
```

User – Marketer:

- Has access to view Event, Participant and Marketer
- Has access to Insert, update and Delete on Marketer only.

```
event_management_sys=# \c postgres
You are now connected to database "postgres" as user "postgres".
postgres=# \c event_management_sys marketer1
Password for user marketer1:
You are now connected to database "event_management_sys" as user "marketer1".
event_management_sys=> delete from Event where Event_ID = 'EID001';
ERROR:  permission denied for table event
event_management_sys=> select * from Participant_;
 participant_id | age | sex |  f_name   |  l_name  |         mail_id          |    username     | password_ | marketer_id | team_id
----------------+-----+-----+-----------+----------+--------------------------+-----------------+-----------+-------------+---------
 PRT001         |  23 | M   | Rishab    | Kumar    | rishabkumar@kjsdv.com    | rishab29        | rishab123 | MAR001      |
 PRT002         |  25 | F   | Deepa     | Girish   | deepagirish@avdva.com    | deepa86         | deepa123  | MAR002      |
 PRT003         |  33 | F   | Nidhi     | Burman   | nidhiburman@ersfg.com    | nidhirokz34     | nidhi123  | MAR003      | TEM002
 PRT004         |  22 | M   | Kaushal   | Reddy    | kaushalreddy@ersfg.com   | kaushalbling    | kaushal123| MAR002      | TEM001
 PRT005         |  21 | M   | Janardhan | Chethri  | janardhancheth@ekffg.com | janardhanbling2 | jannu123  | MAR002      | TEM001
 PRT006         |  29 | F   | Pooja     | Pattekar | poojapattekar@erchg.com  | poojamallu      | pooja123  | MAR003      | TEM003
(6 rows)


event_management_sys=> select * from Team;
ERROR:  permission denied for table team
```

```
event_management_sys=> insert into Marketer(Marketer_ID,Username,Password_) values ('MAR004','Peter','peterg');
INSERT 0 1
```

```
event_management_sys=> select * from Marketer;
 marketer_id | username | password_  | referal_code
-------------+----------+------------+--------------
 MAR001      | John     | john123    | REF001
 MAR002      | Joe      | joe123     | REF002
 MAR003      | Michael  | Michael123 | REF003
 MAR004      | Peter    | peterg     |
(4 rows)


event_management_sys=> delete from Marketer where username = 'Peter';
DELETE 1
event_management_sys=> select * from Marketer;
 marketer_id | username | password_  | referal_code
-------------+----------+------------+--------------
 MAR001      | John     | john123    | REF001
 MAR002      | Joe      | joe123     | REF002
 MAR003      | Michael  | Michael123 | REF003
(3 rows)
```

User – Participant:

- Has access to edit privileges to team, payment_details and register.
- Has access to only View privileges in Event table.

```
event_management_sys=# \c event_management_sys participant1
Password for user participant1:
You are now connected to database "event_management_sys" as user "participant1".
event_management_sys=>
event_management_sys=> select * from Organizer;
ERROR:  permission denied for table organizer
event_management_sys=>
event_management_sys=> select * from Payment_Details;
 payment_id | payment_method | participant_id
------------+----------------+----------------
 PYM001     | Card           | PRT001
 PYM002     | UPI            | PRT002
 PYM003     | Cash           | PRT003
 PYM004     | UPI            | PRT004
 PYM005     | Card           | PRT005
 PYM006     | Cash           | PRT006
(6 rows)
```

Concurrencies:

1) Read Committed Isolation Level: A statement can only see rows committed before it began.

• Open two terminals and connect to the Event Management database on both.

• The first select statement is used to view the Team before the insert statement.

• Insert values into the Team on Terminal 2

• Before commit statement is executed in Terminal 2, we see that the changes aren't reflected (second select statement in Terminal 1)

• After commit statement is executed in Terminal 2, we can see that the changes have now been reflected (third select statement in Terminal 1)

```
event_management_sys=# begin
event_management_sys-# ;
BEGIN
event_management_sys=*# select * from Team;
 team_id |  team_name   | team_size
---------+--------------+-----------
 TEM001  | The Insiders |         2
 TEM002  | Loner        |         1
 TEM003  | FC Barcelona |         1
(3 rows)


event_management_sys=*# select * from Team;
 team_id |  team_name   | team_size
---------+--------------+-----------
 TEM001  | The Insiders |         2
 TEM002  | Loner        |         1
 TEM003  | FC Barcelona |         1
(3 rows)


event_management_sys=*# select * from Team;
 team_id |  team_name   | team_size
---------+--------------+-----------
 TEM001  | The Insiders |         2
 TEM002  | Loner        |         1
 TEM003  | FC Barcelona |         1
 TEM005  | The Boys     |         4
(4 rows)
```

```
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# \c event_management_sys
You are now connected to database "event_management_sys" as user "postgres".
event_management_sys=# begin;
BEGIN
event_management_sys=*# insert into Team(Team_ID,Team_Name,Team_Size) values ('TEM005','The Boys',4);
INSERT 0 1
event_management_sys=*# commit;
COMMIT
event_management_sys=#
```

2) Repeatable Read Isolation Level: All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction

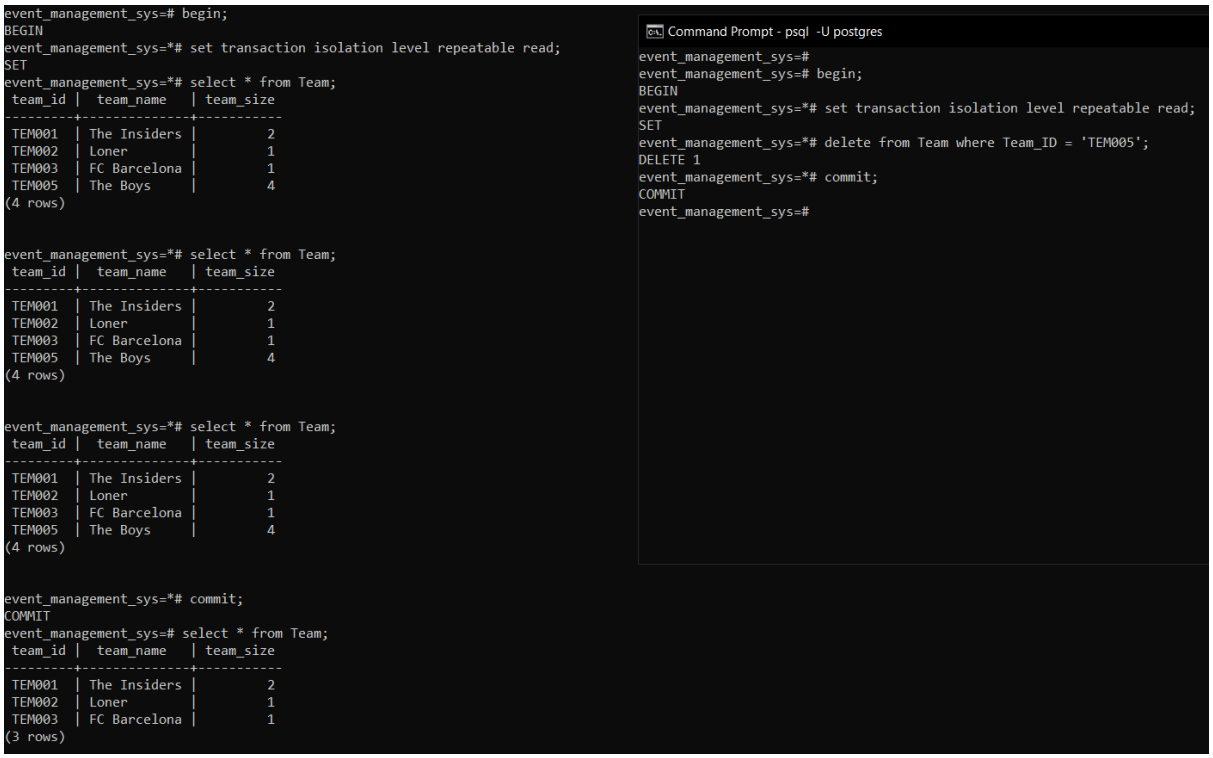
• Open two terminals and connect to the Event Management database on both.

• Set Transaction Isolation Level to Repeatable Read.

• The first select statement in Terminal 1 shows the Team after the delete statement is executed in Terminal 2. (No change reflected)

• The second select statement in Terminal 1 shows the Team after commit statement is executed in Terminal 2. (No change reflected)

• The third select statement shows the Team after the commit statement is executed on Terminal 1. Here we see that the delete operation has finally been reflected.



3)Serializable Isolation Level: All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction. If there is a read/write dependency among the concurrent transactions, one of them will be rolled back with a serialization_failure SQLSTATE.

• Open two terminals and connect to the Event Management database on both.

• Set Transaction Isolation Level to Serializable.

• In Terminal 1 we display the Event Price in which the total cost was GREATER than 300.

• In Terminal 2 we display the Event Price in which the total cost was LESS than 300.

• Then, in Terminal 1 we insert a new row into Event table having total cost LESS than 1000.

• Concurrently, in Terminal 2 we insert a new row into Event table having total cost MORE than 300.
• Thus, a read/write dependency has been introduced in both transactions.

• The commit statement is then executed on both terminals.

• The transaction was ended in Terminal 1 successfully but Terminal 2 gave a serialization error since the access couldn't be serialized due to read/write dependencies among the concurrent transactions.

TERMINAL 1:

```
event_management_sys=# begin;
BEGIN
event_management_sys=*# set transaction isolation level serializable;
SET
event_management_sys=*# select * from Event where Event_Price > 300;
     street      |   city    | pincode | event_id | start_date |  end_date  |                 event_description                 | event_price | event_capacity |    event_category    | organizer_id
-----------------+-----------+---------+----------+------------+------------+---------------------------------------------------+-------------+----------------+----------------------+--------------
 MG road         | Bangalore |  560003 | EID001   | 2021-12-02 | 2021-12-03 | Most Interesting Person in the Room-Kenny Sebastian |         450 |             70 | Comedy               | ORG001
 Gulab Jamun Road | Kolkata   |  560098 | EID002   | 2021-11-09 | 2021-11-10 | Inside - Bo Burnham                               |         500 |             90 | Musical/Comedy       | ORG001
 Bhandra West    | Mumbai    |  560007 | EID003   | 2021-12-11 | 2021-12-13 | Smooth Play                                       |         650 |             90 | Table Tennis         | ORG001
 Gandhinagar     | Delhi     |  560069 | EID004   | 2021-11-15 | 2021-11-19 | Hamilton                                          |         600 |            100 | Open Theater Musical | ORG002
(4 rows)


event_management_sys=*# insert into Event values('VIP Road','Chennai',538402,'EID007','10/11/2021','11/11/2021','Laser Tag',200,50,'Arcade Game','ORG003');
INSERT 0 1
event_management_sys=*# commit;
COMMIT
event_management_sys=#
```

TERMINAL 2:

```
event_management_sys=# begin;
BEGIN
event_management_sys=*# set transaction isolation level serializable;
SET
event_management_sys=*# select * from Event where Event_Price < 300;
     street      |   city    | pincode | event_id | start_date |  end_date  | event_description | event_price | event_capacity | event_category | organizer_id
-----------------+-----------+---------+----------+------------+------------+-------------------+-------------+----------------+----------------+--------------
 100 ft Ring Road | Bangalore |  560010 | EID005   | 2022-01-16 | 2021-01-16 | Benny Dayal Live  |         250 |            500 | Live Concert   | ORG002
 150 ft Ring Road | Bangalore |  560047 | EID006   | 2022-02-20 | 2021-02-22 | Game of Hops      |          50 |             30 | Hopscotch      | ORG003
(2 rows)


event_management_sys=*# insert into Event values('VIP2 Road','Vishakapatnam',577852,'EID008','10/12/2021','11/12/2021','Code Masters',400,90,'Hackathon','ORG002');
INSERT 0 1
event_management_sys=*# commit;
ERROR:  could not serialize access due to read/write dependencies among transactions
DETAIL:  Reason code: Canceled on identification as a pivot, during commit attempt.
HINT:  The transaction might succeed if retried.
event_management_sys=#
```

Individual Contributions:

- K K Tarun Kumar (PES1UG19CS222) – Database Access Privilege Levels, Performance Analysis. (4 Hours)
- Koushik Varma Mandapati (PES1UG19CS230) – Concurrency control, simple queries. (4 Hours)
- Kosaraju Bhargava Krishna (PES1UG19CS225)- Complex Queries and Report writeup. (4 Hours)