

DBMS

Assignment - 3 Report

TEAM ID: 8

TOPIC: OLYMPIC DATABASE

TEAM MEMBERS:

- 1)PRAJWAL KAMATH K - PES1UG19CS337
- 2)PRANAV RAJNISH - PES1UG19CS344
- 3)RAGHAVENDRA L - PES1UG19CS366

QUERY STATEMENTS:

Simple Queries:

1)Display the coach of Athlete “Mercedes Foster”

```
SELECT coach_first_name,coach_last_name
FROM coach,coaches,competitor
WHERE coach.coach_id=coaches.coach_id AND
competitor.competitor_id=coaches.competitor_id AND
competitor.firstname='Mercedes' AND
competitor.lastname='Foster';
```

```
coach_first_name | coach_last_name
-----+-----
Catherine        | Cook
(1 row)
```

2)Display all the events that were held in the 2010 winter Olympics

```
SELECT event.event_name
FROM event, olympic
WHERE event.olympiad_id = olympic.olympiad_id AND
olympic.year_ = 2010 AND olympic.season = 'Winter';
```

```
event_name
-----
Men 100 meter Archery
Women 100 meter Archery
(2 rows)
```

3)For each competitor, list their full names and the total number of races the competitor was in.

```
SELECT competitor.firstname, competitor.lastname,  
COUNT(competitor_event.competitor_id)  
FROM competitor LEFT OUTER JOIN competitor_event ON  
competitor.competitor_id = competitor_event.competitor_id  
GROUP BY competitor.firstname,competitor.lastname;
```

firstname	lastname	count
Marcia	Hardwick	1
Riley	George	0
Samatha	Gardner	2
James	Pinkard	2
Clarence	Rodriguez	0
James	Noonan	1
Mercedes	Foster	2
Patricia	Kowell	2

(8 rows)

4)List the competitor countries in the event men's Basketball in alphabetical order in the 2012 London Olympics

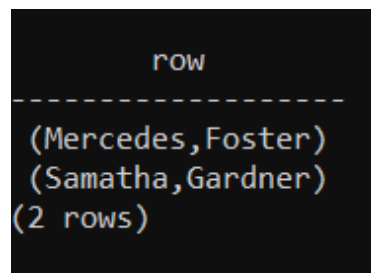
```
SELECT DISTINCT country.country_name  
FROM country, competitor, competitor_event  
WHERE country.country_id = competitor.country_id AND  
competitor.competitor_id = competitor_event.competitor_id  
AND competitor_event.event_id = 1  
ORDER BY country.country_name ASC;
```

country_name
Australia
Greece

(2 rows)

5)List the full names of all competitors in the 2012 London Olympics from Australia.

```
SELECT DISTINCT (competitor.firstname, competitor.lastname)
FROM competitor, competitor_event, event, olympic
WHERE competitor.competitor_id =competitor_event.competitor_id
AND competitor_event.event_id = event.event_id
AND event.olympiad_id = olympic.olympiad_id
AND competitor.country_id = 4
AND olympic.olympiad_id = 1;  -- olympiad id 1 is london, uk 2012
```



```
row
-----
(Mercedes,Foster)
(Samatha,Gardner)
(2 rows)
```

COMPLEX QUERIES:

1)For each country, display the country name and the total number of competitors,including those countries that have no competitors, in descending order of the number of competitors.

```
SELECT DISTINCT country.country_name,
COUNT(competitor_event.competitor_id)
FROM country LEFT OUTER JOIN competitor ON
country.country_id = competitor.country_id
LEFT OUTER JOIN competitor_event ON
competitor.competitor_id = competitor_event.competitor_id
GROUP BY country.country_name
ORDER BY COUNT(competitor_event.competitor_id) DESC;
```

country_name	count
Australia	4
Greece	3
Philippines	3
Africa	0
China	0
Japan	0
South Korea	0
United Kingdom	0
(8 rows)	

2) Count the number of competitors who were from the Philippines in every Olympics held since 1999.

```
SELECT olympic.year_, COUNT(competitor.competitor_id)
FROM competitor LEFT OUTER JOIN competitor_event ON
competitor.competitor_id = competitor_event.competitor_id
LEFT OUTER JOIN event ON
competitor_event.event_id = event.event_id
LEFT OUTER JOIN olympic ON
event.olympiad_id = olympic.olympiad_id
WHERE olympic.year_ >= 1999 AND competitor.country_id = 7
GROUP BY olympic.year_;
```

year_	count
2008	2
2012	1
(2 rows)	

3)List all the competitors who have competed in more than 2 events in any Olympics since 1999.

```
SELECT competitor.competitor_id,competitor.firstname,
competitor.lastname
FROM competitor,competitor_event,event,olympic
WHERE competitor.competitor_id =competitor_event.competitor_id
AND competitor_event.event_id = event.event_id
AND event.olympiad_id = olympic.olympiad_id
AND olympic.year_ >= 1999
GROUP BY competitor.competitor_id, competitor.firstname,
competitor.lastname
HAVING COUNT(competitor_event.competitor_id) >= 2;
```

competitor_id	firstname	lastname
2	Samatha	Gardner
5	James	Pinkard
1	Mercedes	Foster
8	Patricia	Kowell

(4 rows)

4)List All the Athletes with no Medals

```
SELECT competitor.firstname,competitor.lastname
FROM competitor
WHERE competitor.competitor_id NOT IN
(
    SELECT competitor.competitor_id
    FROM competitor,competitor_event
    WHERE
    competitor.competitor_id = competitor_event.competitor_id
    AND competitor_event.medal_id IN (2,3,4)
);
```

firstname	lastname
Riley	George
Clarence	Rodriguez
James	Noonan

(3 rows)

5) Display the results of the men's 1500 freestyle event in the 2008 Beijing Olympics.

```
SELECT competitor.competitor_id, competitor.firstname,
competitor.lastname, medal.medal_name, country.country_name
FROM competitor, competitor_event, event, medal, country
WHERE competitor.competitor_id = competitor_event.competitor_id
AND competitor_event.event_id = event.event_id
AND event.event_id = 7
AND competitor_event.medal_id = medal.medal_id
AND competitor.country_id = country.country_id;
```

competitor_id	firstname	lastname	medal_name	country_name
2	Samatha	Gardner	None	Australia
5	James	Pinkard	Gold	Philippines

(2 rows)

6) Lists the total number of gold medals won by each country.

```
SELECT country.country_name,
COUNT(competitor_event.medal_id)
FROM country, competitor, competitor_event
WHERE country.country_id = competitor.country_id
AND competitor.competitor_id = competitor_event.competitor_id
AND competitor_event.medal_id = 2
GROUP BY country.country_name;
```

country_name	count
Australia	1
Greece	1
Philippines	2

(3 rows)

QUERY EXECUTION PLAN

The execution plan shows how the table(s) referenced by the statement will be scanned — by plain sequential scan, index scan, etc. — and if multiple tables are referenced, what join algorithms will be used to bring together the required rows from each input table.

The ANALYZE option causes the statement to be actually executed, not only planned. The total elapsed time expended within each plan node (in milliseconds) and total number of rows it actually returned are added to the display. This is useful for seeing whether the planner's estimates are close to reality.

If use an indexable attribute(eg. primary key) in the where clause of our queries, we can obtain the result much faster as we can do an index scan instead of a sequential scan.

The planning time is the time postgres takes to make the query execution plan and the execution time is the time taken to execute this plan.

Let us try to analyze this query to find coach for a competitor, based on name.

```
olympic_database=# EXPLAIN ANALYZE SELECT coach_first_name,coach_last_name
olympic_database=# FROM coach,coaches,competitor
olympic_database=# WHERE coach.coach_id=coaches.coach_id AND
olympic_database=#      competitor.competitor_id=coaches.competitor_id AND
olympic_database=#      competitor.firstname='Mercedes' AND competitor.lastname='Foster';
               QUERY PLAN
-----
Nested Loop  (cost=25.26..54.18 rows=5 width=116) (actual time=0.085..0.088 rows=1 loops=1)
->  Nested Loop  (cost=25.11..53.24 rows=5 width=4) (actual time=0.069..0.072 rows=1 loops=1)
    -> Seq Scan on competitor  (cost=0.00..17.35 rows=1 width=4) (actual time=0.042..0.044 rows=1 loops=1)
        Filter: (((firstname)::text = 'Mercedes'::text) AND ((lastname)::text = 'Foster'::text))
        Rows Removed by Filter: 7
    -> Bitmap Heap Scan on coaches  (cost=25.11..35.78 rows=11 width=8) (actual time=0.015..0.015 rows=1 loops=1)
        Recheck Cond: (competitor_id = competitor.competitor_id)
        Heap Blocks: exact=1
        -> Bitmap Index Scan on coaches_pk  (cost=0.00..25.11 rows=11 width=0) (actual time=0.010..0.010 rows=1 loops=1)
            Index Cond: (competitor_id = competitor.competitor_id)
-> Index Scan using coach_pk on coach  (cost=0.15..0.19 rows=1 width=120) (actual time=0.015..0.015 rows=1 loops=1)
    Index Cond: (coach_id = coaches.coach_id)
Planning Time: 1.819 ms
Execution Time: 0.202 ms
(14 rows)
```

For each node or level, the values in the brackets give us the expected values and the second set gives us the actual metrics.

For comparing the name, a sequential scan is done on competitor, then a bitmap heap scan on coaches and a Bitmap Index Scan can be done for comparing competitor_id as it is an indexable attribute since it is a primary key. The same is true for checking coach_id.

For the next query, we are using a nested query to find the desired output.

```
olympic_database=# EXPLAIN ANALYZE SELECT competitor.firstname,competitor.lastname
olympic_database=# FROM competitor
olympic_database=# WHERE competitor.competitor_id NOT IN(
olympic_database(# SELECT competitor.competitor_id
olympic_database(# FROM competitor,competitor_event
olympic_database(# WHERE competitor.competitor_id = competitor_event.competitor_id
olympic_database(# AND competitor_event.medal_id IN (2,3,4)
olympic_database(# );
                                QUERY PLAN
-----
Seq Scan on competitor (cost=56.61..72.73 rows=245 width=116) (actual time=0.141..0.144 rows=3 loops=1)
  Filter: (NOT (hashed SubPlan 1))
  Rows Removed by Filter: 5
  SubPlan 1
    -> Hash Join (cost=21.02..56.54 rows=28 width=4) (actual time=0.074..0.078 rows=6 loops=1)
      Hash Cond: (competitor_event.competitor_id = competitor_1.competitor_id)
      -> Seq Scan on competitor_event (cost=0.00..35.44 rows=28 width=4) (actual time=0.025..0.028 rows=6 loops=1)
        Filter: (medal_id = ANY ('{2,3,4}':integer[]))
        Rows Removed by Filter: 4
      -> Hash (cost=14.90..14.90 rows=490 width=4) (actual time=0.031..0.031 rows=8 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 9kB
        -> Seq Scan on competitor competitor_1 (cost=0.00..14.90 rows=490 width=4) (actual time=0.012..0.013 rows=8 loops=1)
Planning Time: 0.854 ms
Execution Time: 0.236 ms
(14 rows)
```

For this query, we get a sub plan since we have a nested query. We have a hash join since we are referencing two tables on the condition competitor_event_id = competitor_id.

We then do a sequential scan on competitor event and filter those events that meet our condition.

Similarly in the next query, we have a lot of hash joins on the respective hash conditions.

```
olympic_database=# EXPLAIN ANALYZE SELECT competitor.competitor_id,
olympic_database=# competitor.firstname,
olympic_database=# competitor.lastname
olympic_database=# FROM competitor,
olympic_database=# competitor_event,
olympic_database=# event,
olympic_database=# olympic
olympic_database=# WHERE competitor.competitor_id = competitor_event.competitor_id AND
olympic_database=# competitor_event.event_id = event.event_id AND
olympic_database=# event.olympiad_id = olympic.olympiad_id AND
olympic_database=# olympic.year_ >= 1999;
                                QUERY PLAN
-----
Hash Join (cost=52.59..92.59 rows=618 width=120) (actual time=0.164..0.175 rows=10 loops=1)
  Hash Cond: (competitor_event.competitor_id = competitor.competitor_id)
  -> Hash Join (cost=31.56..69.93 rows=618 width=4) (actual time=0.125..0.133 rows=10 loops=1)
    Hash Cond: (event.olympiad_id = olympic.olympiad_id)
    -> Hash Join (cost=14.72..48.18 rows=1850 width=8) (actual time=0.074..0.080 rows=10 loops=1)
      Hash Cond: (competitor_event.event_id = event.event_id)
      -> Seq Scan on competitor_event (cost=0.00..28.50 rows=1850 width=8) (actual time=0.013..0.015 rows=10 loops=1)
      -> Hash (cost=12.10..12.10 rows=210 width=8) (actual time=0.030..0.030 rows=9 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 9kB
        -> Seq Scan on event (cost=0.00..12.10 rows=210 width=8) (actual time=0.017..0.018 rows=9 loops=1)
    -> Hash (cost=15.13..15.13 rows=137 width=4) (actual time=0.032..0.032 rows=5 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 9kB
      -> Seq Scan on olympic (cost=0.00..15.13 rows=137 width=4) (actual time=0.021..0.022 rows=5 loops=1)
        Filter: (year_ >= 1999)
        Rows Removed by Filter: 1
  -> Hash (cost=14.90..14.90 rows=490 width=120) (actual time=0.032..0.032 rows=8 loops=1)
    Buckets: 1024 Batches: 1 Memory Usage: 9kB
    -> Seq Scan on competitor (cost=0.00..14.90 rows=490 width=120) (actual time=0.023..0.025 rows=8 loops=1)
Planning Time: 1.051 ms
Execution Time: 0.273 ms
(20 rows)
```

We then have to do a sequential scan to find the rows that we are interested in. If we were using an indexable attribute to search the table, the queries would execute a lot faster since an indexable search is much faster than sequential search.

GRANTING ACCESS PRIVILEGES FOR MULTIPLE USERS

Three Users ADMIN, ATHLETE and ORGANIZER are created
ADMIN is created as SUPERUSER with CREATEROLE

```
olympic_database=# create user admin with password '8088' SUPERUSER CREATEDB CREATEROLE;  
CREATE ROLE  
olympic_database=# create user athlete with password '1234' CREATEDB;  
CREATE ROLE  
olympic_database=# create user organizer with password '999' CREATEDB;  
CREATE ROLE
```

ALL Permission is given to ORGANISER on All Tables
SELECT Permission is given to ATHLETE

```
olympic_database=# grant all on all tables in schema "public" to organizer;  
GRANT  
olympic_database=# grant select on COMPETITOR to athlete;  
GRANT
```

ORGANIZER can Perform All Operations on All Tables as Shown below

```
C:\Program Files\PostgreSQL\12\bin>psql -U organizer -d olympic_database  
Password for user organizer:  
psql (12.8)  
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.  
  
olympic_database=> select * from competitor;  
 competitor_id | country_id | firstname | lastname | gender | date_of_birth  
-----+-----+-----+-----+-----+-----  
1 | 4 | Mercedes | Foster | M | 1988-12-18  
2 | 4 | Samatha | Gardner | F | 1984-07-27  
3 | 1 | Riley | George | M | 1952-08-19  
4 | 1 | Clarence | Rodriguez | F | 1986-12-23  
5 | 7 | James | Pinkard | M | 1985-04-08  
6 | 7 | Marcia | Hardwick | F | 1980-07-30  
7 | 2 | James | Noonan | M | 1987-09-25  
8 | 2 | Patricia | Kowell | F | 1986-06-16  
(8 rows)  
  
olympic_database=> INSERT INTO country VALUES (10,'INDIA');  
INSERT 0 1  
olympic_database=> DELETE FROM country where country_id=10;  
DELETE 1  
olympic_database=> select * from country;  
 country_id | country_name  
-----+-----  
1 | United Kingdom  
2 | Greece  
3 | China  
4 | Australia  
5 | South Korea  
6 | Japan  
7 | Philippines  
8 | Africa  
(8 rows)
```

ATHLETE can only perform Select Operations

```
C:\Program Files\PostgreSQL\12\bin>psql -U athlete -d olympic_database
Password for user athlete:
psql (12.8)
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.

olympic_database=> select * from competitor
olympic_database-> ;
 competitor_id | country_id | firstname | lastname | gender | date_of_birth
-----+-----+-----+-----+-----+-----
           1 |          4 | Mercedes | Foster   | M      | 1988-12-18
           2 |          4 | Samatha  | Gardner  | F      | 1984-07-27
           3 |          1 | Riley    | George   | M      | 1952-08-19
           4 |          1 | Clarence | Rodriguez| F      | 1986-12-23
           5 |          7 | James    | Pinkard  | M      | 1985-04-08
           6 |          7 | Marcia   | Hardwick | F      | 1980-07-30
           7 |          2 | James    | Noonan   | M      | 1987-09-25
           8 |          2 | Patricia | Kowell   | F      | 1986-06-16
(8 rows)

olympic_database=> select * from country;
ERROR: permission denied for table country
olympic_database=> DELETE FROM country where country_id=10;
ERROR: permission denied for table country
olympic_database=>
```

Using ADMIN we are creating another User called VISITOR which has select Permission on EVENT table

```
C:\Program Files\PostgreSQL\12\bin>psql -U admin -d olympic_database
Password for user admin:
psql (12.8)
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.

olympic_database=# create user visitor with password 'xyz' CREATEDB;
CREATE ROLE
olympic_database=# grant select on event to visitor;
GRANT
```

The VISITOR can only perform Select Operation on EVENT table

```
C:\Program Files\PostgreSQL\12\bin>psql -U visitor -d olympic_database
Password for user visitor:
psql (12.8)
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.

olympic_database=> select * from event;
 event_id | olympiad_id | category_id | event_name | gender | team | location | start_date | end_date
-----+-----+-----+-----+-----+-----+-----+-----+-----
      1 |          1 |          1 | Men Basketball | M | T | Burnham Park | 2012-03-08 | 2012-03-10
      2 |          1 |          6 | Men 1500 Freestyle event | M | F | Pool Park | 2012-02-10 | 2012-02-12
      3 |          5 |          1 | Women Basketball | F | T | Shrine Court | 1998-04-12 | 1998-04-14
      4 |          4 |          5 | Men Welter Weight Boxing | M | F | Douglas Stadium | 1999-05-14 | 1999-05-16
      5 |          3 |          5 | Men Heavy Weight Boxing | M | F | Pilar Park | 2008-06-16 | 2008-06-18
      6 |          1 |          1 | Women Basketball | F | T | London Bridge Court | 2012-08-18 | 2012-08-20
      7 |          3 |          6 | Men 1500 Freestyle event | M | F | Xiang Pool | 2008-09-20 | 2008-09-22
      8 |          6 |          2 | Men 100 meter Archery | M | F | Sahara Desert | 2010-10-22 | 2010-10-24
      9 |          6 |          2 | Women 100 meter Archery | F | F | Sahara Desert | 2010-11-24 | 2010-11-26
(9 rows)
```

We are Deleting the SUPERUSER ADMIN which created VISITOR

```
C:\Program Files\PostgreSQL\12\bin>psql -U postgres -d olympic_database
Password for user postgres:
psql (12.8)
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.

olympic_database=# DROP USER admin;
DROP ROLE
```

Even After Deleting ADMIN we are still able to perform SELECT operation using VISITOR

```
C:\Program Files\PostgreSQL\12\bin>psql -U visitor -d olympic_database
Password for user visitor:
psql (12.8)
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.

olympic_database=> select * from event;
 event_id | olympiad_id | category_id | event_name | gender | team | location | start_date | end_date
-----+-----+-----+-----+-----+-----+-----+-----+-----
      1 |          1 |          1 | Men Basketball | M | T | Burnham Park | 2012-03-08 | 2012-03-10
      2 |          1 |          6 | Men 1500 Freestyle event | M | F | Pool Park | 2012-02-10 | 2012-02-12
      3 |          5 |          1 | Women Basketball | F | T | Shrine Court | 1998-04-12 | 1998-04-14
      4 |          4 |          5 | Men Welter Weight Boxing | M | F | Douglas Stadium | 1999-05-14 | 1999-05-16
      5 |          3 |          5 | Men Heavy Weight Boxing | M | F | Pilar Park | 2008-06-16 | 2008-06-18
      6 |          1 |          1 | Women Basketball | F | T | London Bridge Court | 2012-08-18 | 2012-08-20
      7 |          3 |          6 | Men 1500 Freestyle event | M | F | Xiang Pool | 2008-09-20 | 2008-09-22
      8 |          6 |          2 | Men 100 meter Archery | M | F | Sahara Desert | 2010-10-22 | 2010-10-24
      9 |          6 |          2 | Women 100 meter Archery | F | F | Sahara Desert | 2010-11-24 | 2010-11-26
(9 rows)

olympic_database=> select * from competitor;
ERROR: permission denied for table competitor
```

Authorization using Views

We have created a View called Athletes_names through which users can only view the names of competitors

```
olympic_database=# create view athletes_names as SELECT firstname,lastname from competitor;  
CREATE VIEW
```

The view created can only access the names of the competitors

```
olympic_database=# select * from athletes_names;  
  firstname | lastname  
-----+-----  
 Mercedes   | Foster  
 Samatha    | Gardner  
 Riley      | George  
 Clarence    | Rodriguez  
 James      | Pinkard  
 Marcia      | Hardwick  
 James      | Noonan  
 Patricia    | Kowell  
(8 rows)
```

```
olympic_database=# select * from competitor;  
competitor_id | country_id | firstname | lastname | gender | date_of_birth  
-----+-----+-----+-----+-----+-----  
          1 |         4 | Mercedes  | Foster   | M      | 1988-12-18  
          2 |         4 | Samatha   | Gardner  | F      | 1984-07-27  
          3 |         1 | Riley     | George   | M      | 1952-08-19  
          4 |         1 | Clarence  | Rodriguez| F      | 1986-12-23  
          5 |         7 | James     | Pinkard  | M      | 1985-04-08  
          6 |         7 | Marcia    | Hardwick | F      | 1980-07-30  
          7 |         2 | James     | Noonan   | M      | 1987-09-25  
          8 |         2 | Patricia  | Kowell   | F      | 1986-06-16  
(8 rows)
```

The original competitor table had all the information related to the Athlete.

TRANSACTIONS

EXAMPLE 1

In terminal 1 we start a transaction and update data i.e. Updating the country name to 'INDIA' in the Country Table

```
olympic_database=# begin;  
BEGIN  
olympic_database=# update country set country_name='INDIA' where country_id=8;  
UPDATE 1  
olympic_database=# select * from country  
olympic_database=# ;  
 country_id | country_name  
-----+-----  
      1 | United Kingdom  
      2 | Greece  
      3 | China  
      4 | Australia  
      5 | South Korea  
      6 | Japan  
      7 | Philippines  
      8 | INDIA  
(8 rows)
```

Terminal1

Now in Terminal 2 we try to access the country table

```
olympic_database=# select * from country;  
 country_id | country_name  
-----+-----  
      1 | United Kingdom  
      2 | Greece  
      3 | China  
      4 | Australia  
      5 | South Korea  
      6 | Japan  
      7 | Philippines  
      8 | Africa  
(8 rows)
```

Terminal 2

After updating the record, the transaction in Terminal-1 is not committed, the select statement in that transaction is available to view that uncommitted change. But the select statement in Terminal-2 is

unable to view the change since the transaction in Terminal1 is still uncommitted .

Now in Terminal1 we end the transaction

```
olympic_database=# begin;
BEGIN
olympic_database=# update country set country_name='INDIA' where country_id=8;
UPDATE 1
olympic_database=# select * from country
olympic_database-## ;
 country_id | country_name
-----+-----
          1 | United Kingdom
          2 | Greece
          3 | China
          4 | Australia
          5 | South Korea
          6 | Japan
          7 | Philippines
          8 | INDIA
(8 rows)

olympic_database=# end transaction;
COMMIT
olympic_database=#
```

Terminal 1

After the transaction in Terminal1 has been committed , the select statement in Terminal2 now can see the updated value. Therefore the select statement works on the snapshot of the database which has the latest committed change without being able to view the un-committed changes.

```
olympic_database=# select * from country;
 country_id | country_name
-----+-----
          1 | United Kingdom
          2 | Greece
          3 | China
          4 | Australia
          5 | South Korea
          6 | Japan
          7 | Philippines
          8 | INDIA
(8 rows)

olympic_database=#
```

EXAMPLE2

In terminal 1 we start transaction and update the country_name which is updated successfully

```
olympic_database=# begin;  
BEGIN  
olympic_database=# update country set country_name='AFRICA' where country_name='NEPAL';  
UPDATE 1  
olympic_database=#
```

Terminal1

In terminal 2 we start transaction simultaneously with respect to terminal 1 and we try to update the same country_name. We observe that update doesn't take place in terminal 2 because it waits for terminal 1 to complete the transaction

```
olympic_database=# begin;  
BEGIN  
olympic_database=# update country set country_name='GERMANY' where country_name='NEPAL';
```

Terminal2

Now in Terminal 1 we make a syntax error . As a result of this the transaction in terminal 1 is not committed as result the update which we had performed earlier is not saved or committed.

```
olympic_database=# update country set country_name='AFRICA' where country_nae='NEPAL';  
ERROR: column "country_nae" does not exist  
LINE 1: update country set country_name='AFRICA' where country_nae='...  
                        ^  
HINT: Perhaps you meant to reference the column "country.country_name".  
olympic_database=# end transaction  
olympic_database=# ;  
ROLLBACK  
olympic_database=#
```

Terminal1

Now as the Transaction in Terminal1 is not saved, the transaction in terminal2 that was waiting for terminal1 to finish performs the update operation.

```
olympic_database=# begin;  
BEGIN  
olympic_database=# update country set country_name='GERMANY' where country_name='NEPAL';  
UPDATE 1  
olympic_database=# end transaction;  
COMMIT  
olympic_database=#
```


Contributions

- 1) Prajwal Kamath K- SQL Queries, Access Privileges, Views
Hours Spent-4
- 2) Pranav Rajnish- SQL Queries, Query Execution Plan
Hours Spent-3
- 3) Raghavendra L -SQL Queries, Transactions
Hours Spent-3