DBMS Assignment – 3

Guhan K

PES1UG19CS171

Hanuraag Ravilla Bhaskaran

PES1UG19CS177

Harshita Vidapanakal

PES1UG19CS185

Demonstrating 5 simple queries

1) select * from Batsman

```
SELECT * FROM Batsman
```

100 %

Results | Messages

	Balls_Played	Scored	Fours	Sixes	Player_ID
1	50	80	4	5	21345
2	70	80	4	5	45123
3	46	77	6	2	67345
4	64	110	7	8	78546
5	29	23	1	0	89765

2) select Match_ID from Match where Win_Method='runs'

```sql
select Match_ID from Match where Win_Method='runs'
```

P-C1RC1D4R (SQL Server 15.0.2000.5 - LAPTOP-C1RC1D4R\Hanuraag Baskaran)

100 %

Results | Messages

	Match_ID
1	1233
2	1234
3	1235
4	1236
5	1239
6	1240

3) select Player_ID from Player where Team_Name='Chennai Super Kings'

```sql
select Player_ID from Player where Team_Name='Chennai Super Kings'
```

100 %

Results | Messages

	Player_ID
1	67345
2	67890

4) select Mnemonic from TEAM where Home_Stadium='Punjab'

```
select Mnemonic from TEAM where Home_Stadium='Punjab'
```

100 %

Results | Messages

	Mnemonic
1	PBKS

5)
```
select Runs_Given from Bowler where Wickets_Taken=0
```

```
select Runs_Given from Bowler where Wickets_Taken=0
```

100 %

Results | Messages

	Runs_Given
1	40
2	16
3	23
4	44

Demonstrating 5 complex queries

1)
```
select avg(Innings.Run_Rate) from Innings
        inner join Match on Innings.Match_ID=Match.Match_ID
```

```
select avg(Innings.Run_Rate) from Innings
inner join Match on Innings.Match_ID=Match.Match_ID
```

%

Results | Messages

(No column name)
7.62

2)
```
select Name, Scored from Player
full outer join Batsman on
Player.Player_ID=Batsman.Player_ID
where Balls_Played>50
```

```
select Name, Scored from Player
full outer join Batsman on Player.Player_ID=Batsman.Player_ID
where Balls_Played>50
```

100 %

Results | Messages

	Name	Scored
1	Rohit Sharma	80
2	Rishabh Pant	110

3)
```
select Coach.Name, Player.Name from Coach
inner join Player on Coach.Team_ID=Player.Team_Name
inner join Bowler on Player.Player_ID=Bowler.Player_ID
where Bowler.Wickets_Taken>1
```

```sql
select Coach.Name, Player.Name from Coach
inner join Player on Coach.Team_ID=Player.Team_Name
inner join Bowler on Player.Player_ID=Bowler.Player_ID
where Bowler.Wickets_Taken>1
```

100 %

Results | Messages

	Name	Name
1	Andrew Mcdonald	Sanju Samson
2	Amol Muzumdar	Sanju Samson
3	Ricky Ponting	Rishabh Pant

4)
```sql
select Team.Name, Match.Won_By from Match
full outer join Played_By on Match.Match_ID=Played_By.Match_ID
full outer join TEAM on Played_By.Match_ID=TEAM.Name
```

SQLQuery1.sql - LA...raag Baskaran (59))*

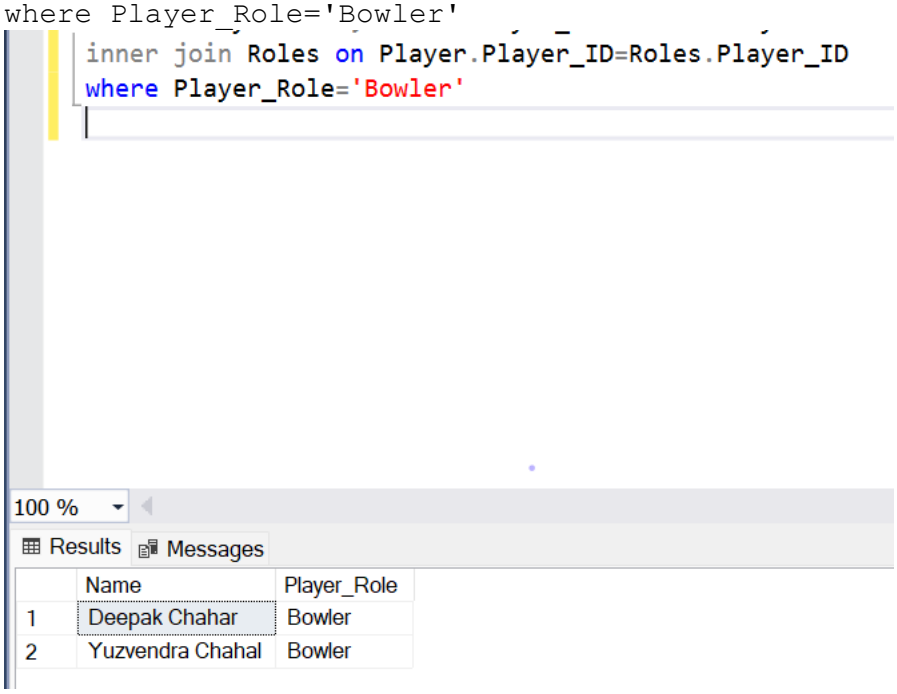
```sql
select Team.Name, Match.Won_By from Match
full outer join Played_By on Match.Match_ID=Played_By.Match_ID
full outer join TEAM on Played_By.Match_ID=TEAM.Name
```

100 %

Results | Messages

	Name	Won_By
5	NULL	10
6	NULL	10
7	NULL	4
8	NULL	4
9	NULL	10
10	NULL	10
11	NULL	6
12	NULL	6
13	NULL	3
14	NULL	3
15	NULL	6
16	NULL	6
17	NULL	13
18	NULL	13
19	NULL	38
20	NULL	38
21	Punjab Kings	NULL
22	Mumbai Indians	NULL
23	Sunrisers Hyderabad	NULL
24	Delhi Capitals	NULL
25	Royal Challengers ...	NULL
26	Kolkata Knight Ride...	NULL

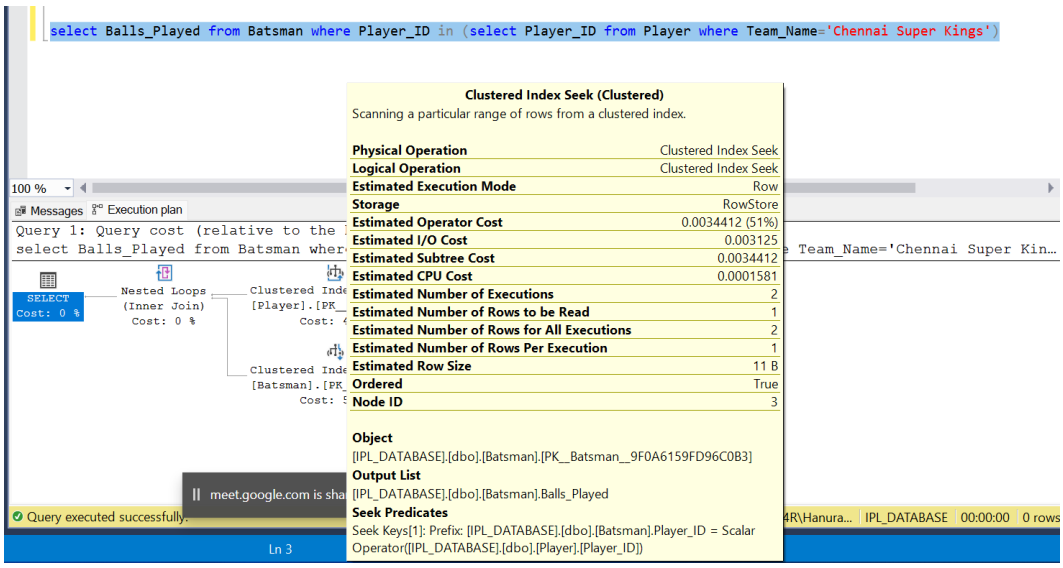
5) `select Player.Name, Roles.Player_Role from Player inner join Roles on Player.Player_ID=Roles.Player_ID where Player_Role='Bowler'`



Query Execution Plans using Nested Queries

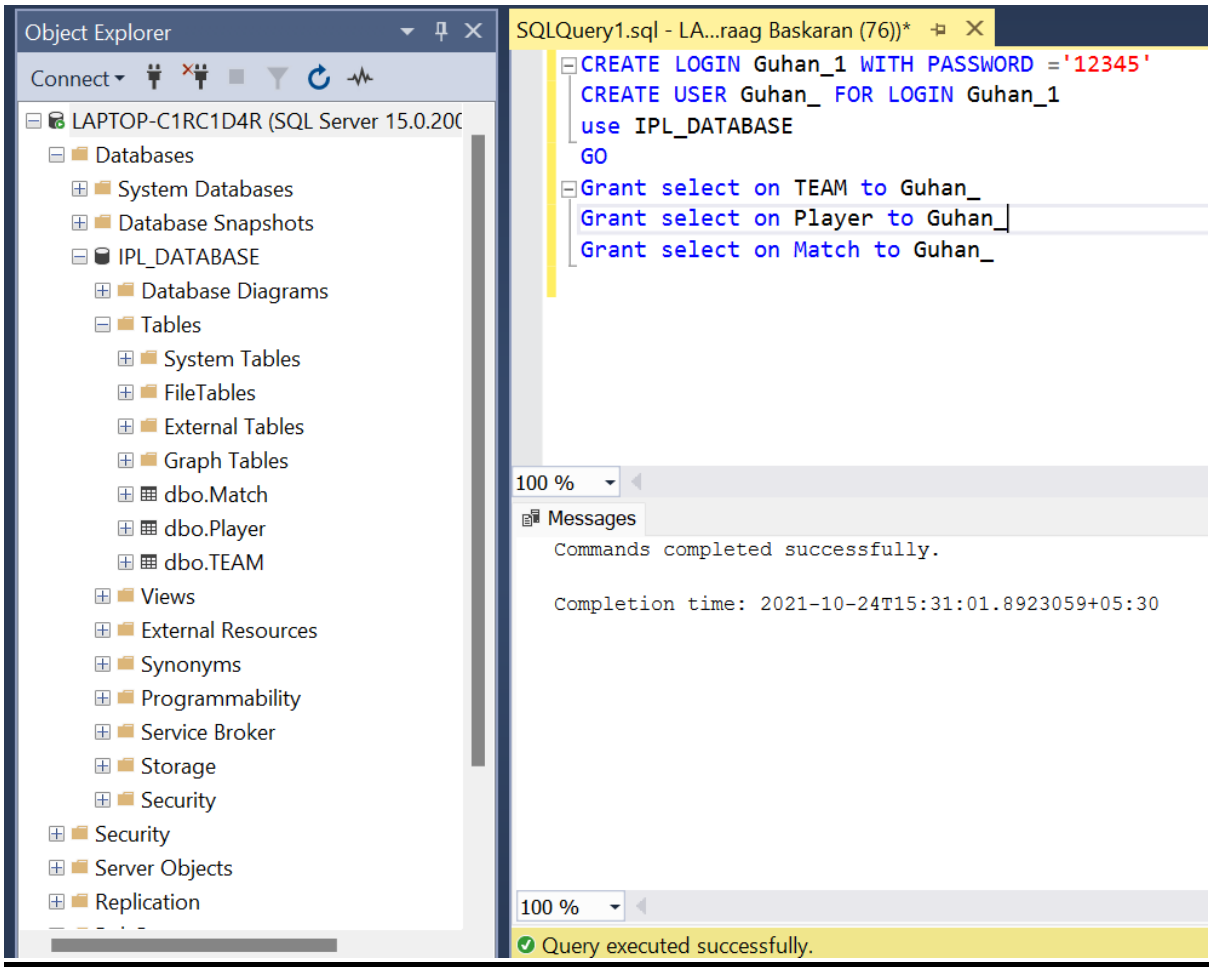
We shall be using the following nested query for creating a Query Execution plan:
`>select Balls_Played from Batsman where Player_ID in (select Player_ID from Player where Team_Name='Chennai Super Kings')`

Using SQL Server, we can get readymade execution plans for the given query, by highlighting the query and pressing Ctrl+L, we get the estimated execution plan



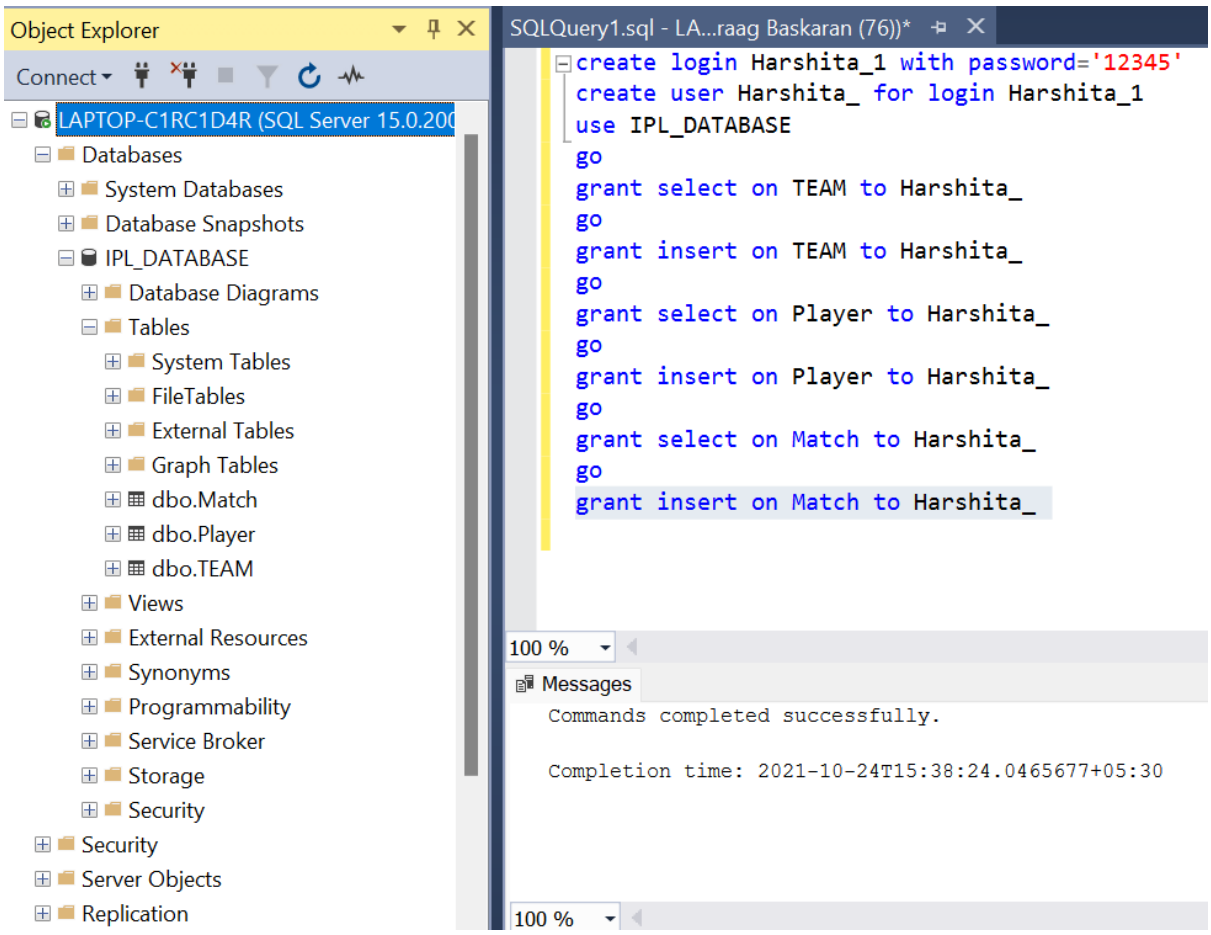
Users with Different Logins

```
CREATE LOGIN Guhan_1 WITH PASSWORD ='12345'
CREATE USER Guhan_ FOR LOGIN Guhan_1
use IPL_DATABASE
GO
Grant select on TEAM to Guhan_
Grant select on Player to Guhan_
Grant select on Match to Guhan_
```




```
create login Harshita_1 with password='12345'
create user Harshita_ for login Harshita_1
use IPL_DATABASE
go
grant select on TEAM to Harshita_
go
grant insert on TEAM to Harshita_
go
grant select on Player to Harshita_
go
grant insert on Player to Harshita_
go
```

```
grant select on Match to Harshita_
go
grant insert on Match to Harshita
```



Concurrent Transactions

By default SQL Server has inbuilt protection against concurrent transactions, by using read committed as the default state to protect against concurrency errors.

We will demonstrate this by simulating a dirty read using two scripts. The first one is as follows;

Script 1:

```
select count(distinct Stadium_ID) BeforeTransactionStarts from Match
where Toss_Decision='Batting'

begin transaction;

Update Match
set Toss_Decision='Bowling'

select count(distinct Stadium_Id) DuringTransaction from Match
where Toss_Decision='Batting'
```

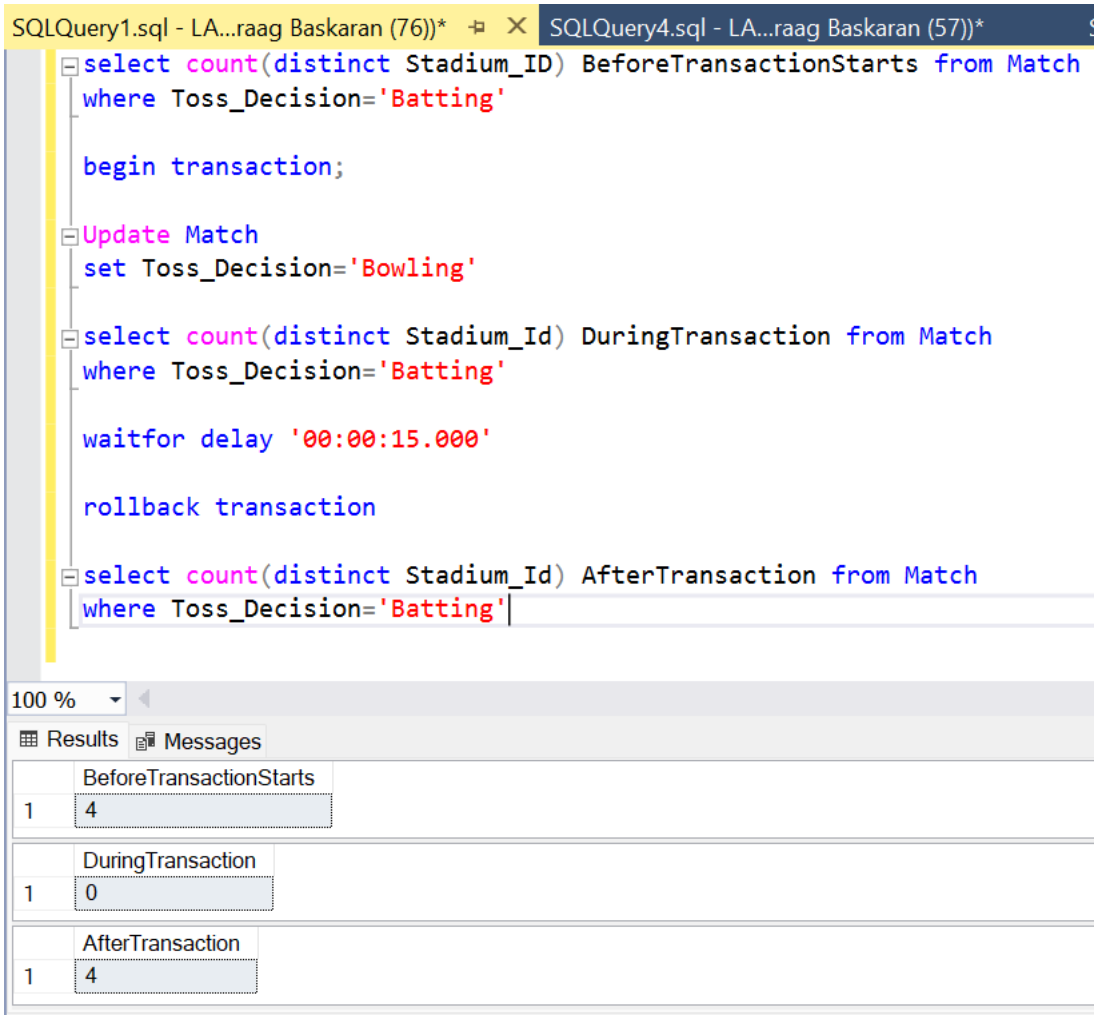
```
waitfor delay '00:00:15.000'

rollback transaction

select count(distinct Stadium_Id) AfterTransaction from Match
where Toss_Decision='Batting'
```

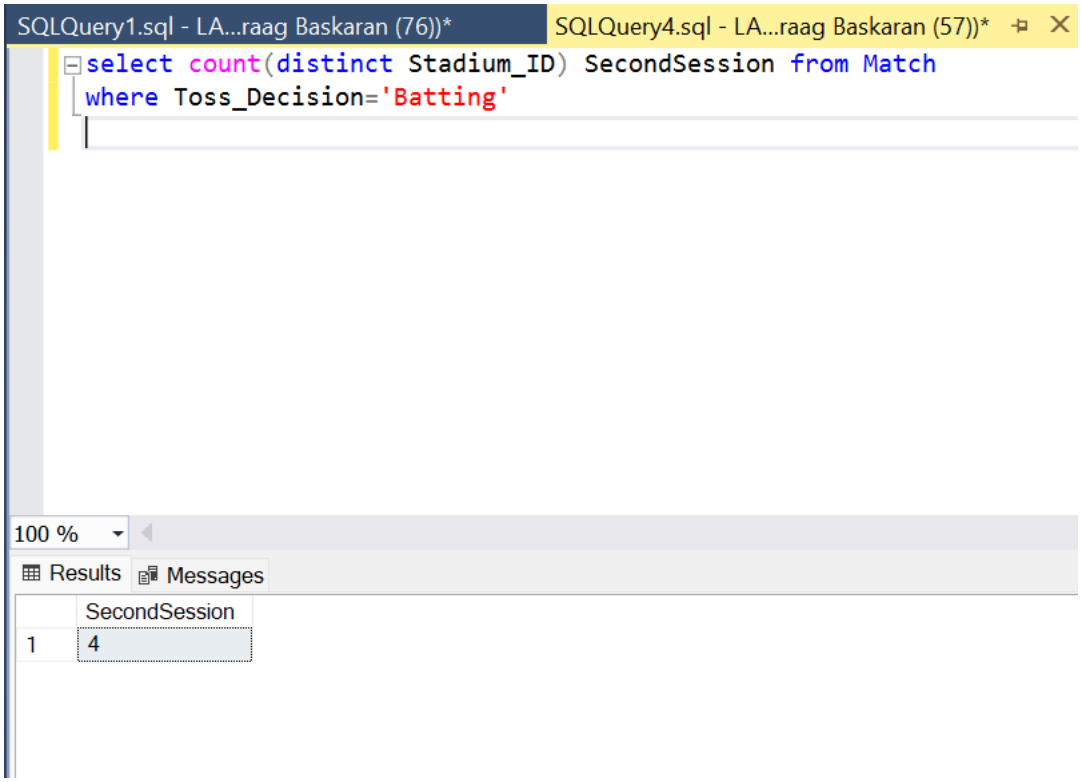
During the wait time of the first script, we will run this second script

Script 2:

```
select count(distinct Stadium_ID) SecondSession from Match
where Toss_Decision='Batting'
```

With the default concurrency protection, the second script will only run after the first script has been done executing, even though it was executed separately during the first scripts execution.





Contribution:

Guhan – Performance Analysis, Nested Queries, Multiple User access

Hanuraag – Concurrent Transactions, Nested Queries, Performance Analysis, Multiple User Access

Harshita – Simple Queries, Complex Queries, Multiple User Access