

# DBMS Assignment – 3

Guhan K

PES1UG19CS171

Hanuraag Ravilla Bhaskaran

PES1UG19CS177

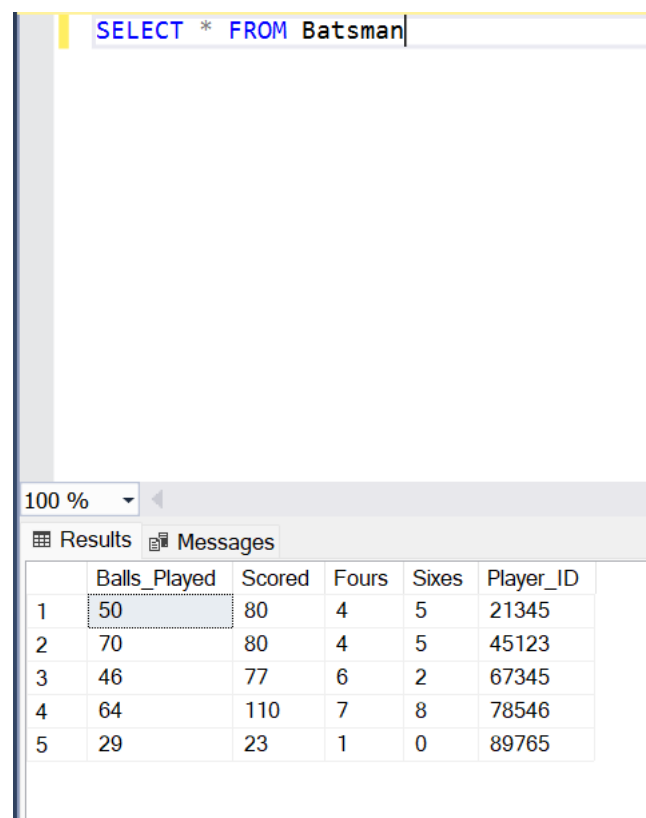
Harshita Vidapanakal

PES1UG19CS185

---

## Demonstrating 5 simple queries

1) select \* from Batsman



The screenshot shows a SQL query execution interface. At the top, the query 'SELECT \* FROM Batsman' is entered in a text box. Below the text box, there is a 'Results' tab and a 'Messages' tab. The 'Results' tab is active, displaying a table with 5 rows and 6 columns. The columns are 'Balls\_Played', 'Scored', 'Fours', 'Sixes', and 'Player\_ID'. The rows contain the following data:

	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'

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'

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
```

```

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) 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))* X
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

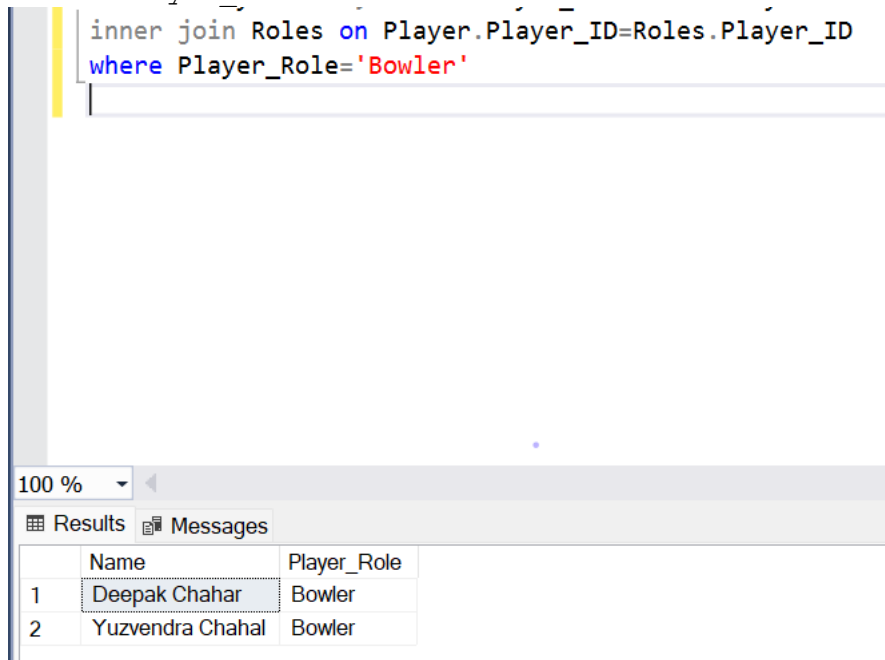
```

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'`



The screenshot shows a SQL Server query window with the following query:

```
inner join Roles on Player.Player_ID=Roles.Player_ID  
where Player_Role='Bowler'
```

The results pane displays a table with two rows:

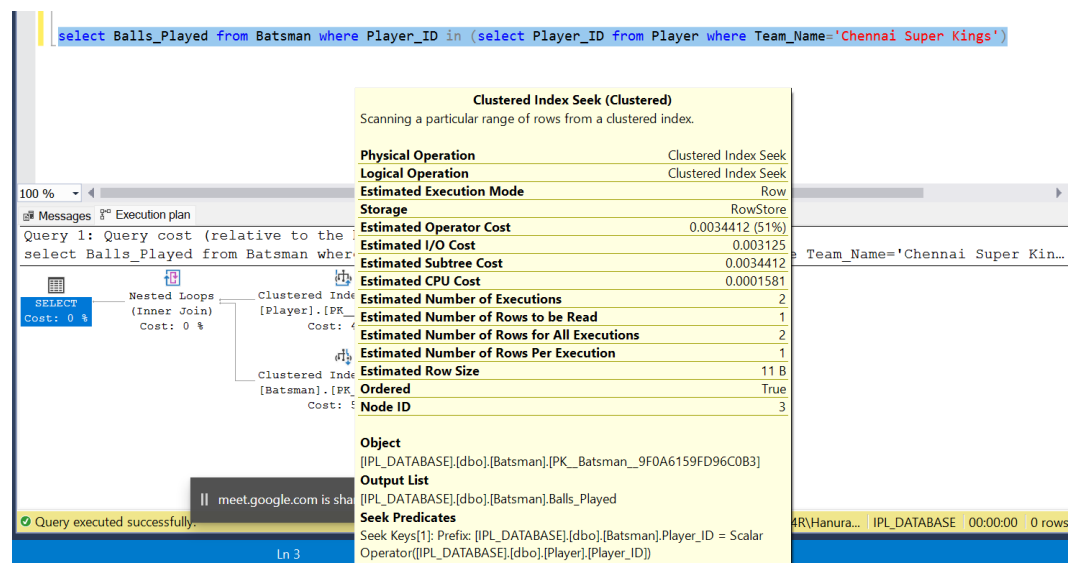
	Name	Player_Role
1	Deepak Chahar	Bowler
2	Yuzvendra Chahal	Bowler

## 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



The screenshot shows the SQL Server query window with the following query:

```
select Balls_Played from Batsman where Player_ID in (select Player_ID from Player where Team_Name='Chennai Super Kings')
```

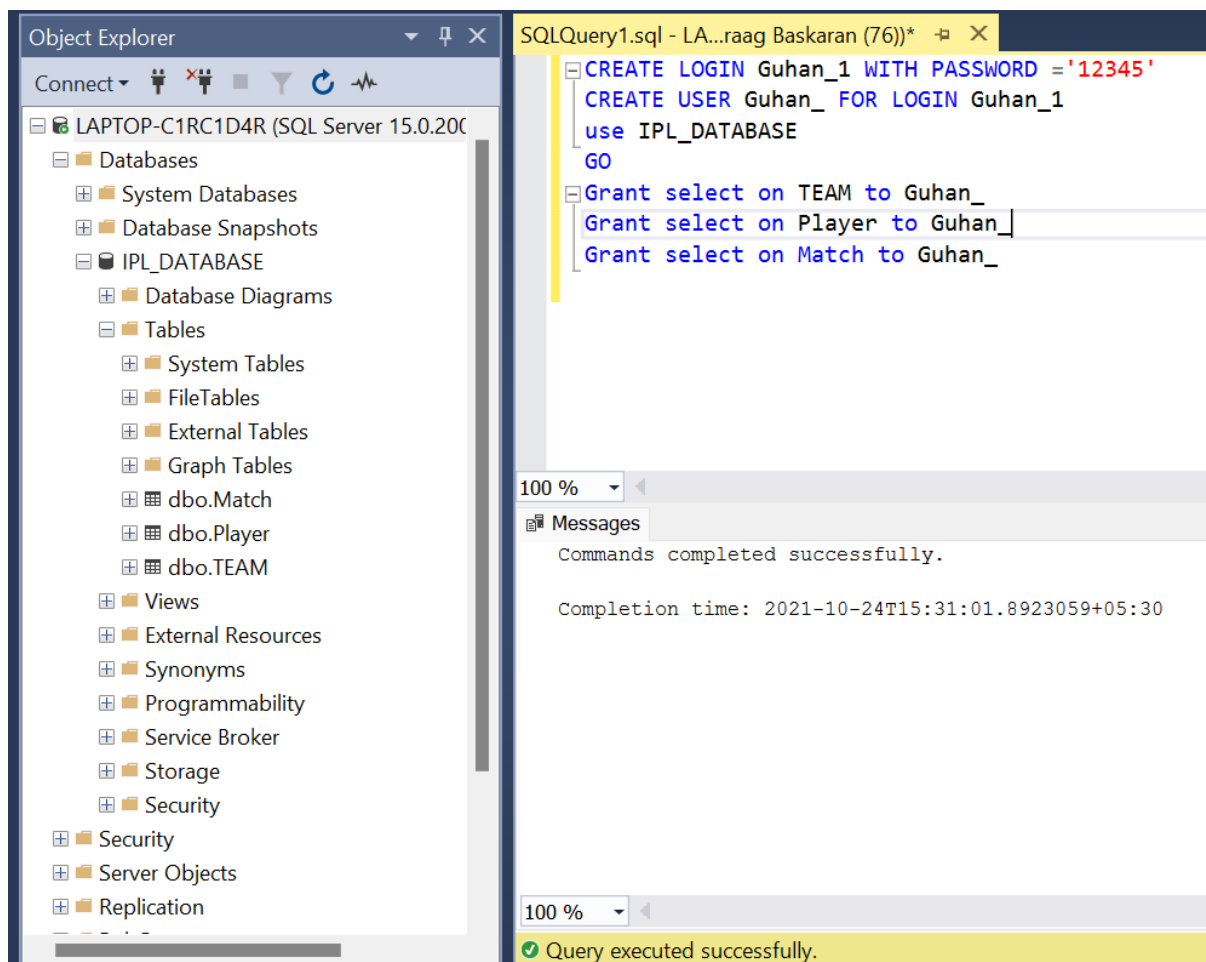
The execution plan is displayed, showing a nested query structure. The main query is a **SELECT** statement with a cost of 0%. It involves a **Nested Loops (Inner Join)** operation. The inner query is a **SELECT** statement with a cost of 0%, which is a **Clustered Index Seek (Clustered)** operation. The execution plan details are as follows:

Operation	Cost	Estimated Number of Executions	Estimated Number of Rows to be Read	Estimated Number of Rows for All Executions	Estimated Number of Rows Per Execution	Estimated Row Size	Ordered	Node ID
Clustered Index Seek (Clustered)	0.0034412 (51%)	2	1	2	1	11 B	True	3
Storage	0.003125							
Estimated Operator Cost	0.0034412 (51%)							
Estimated I/O Cost	0.003125							
Estimated Subtree Cost	0.0034412							
Estimated CPU Cost	0.0001581							
Estimated Number of Executions		2						
Estimated Number of Rows to be Read		1						
Estimated Number of Rows for All Executions		2						
Estimated Number of Rows Per Execution		1						
Estimated Row Size		11 B						
Ordered		True						
Node ID		3						

The execution plan also shows the **Object** as `[IPL_DATABASE].[dbo].[Batsman].[PK_Batsman__9F0A6159FD96C0B3]` and the **Output List** as `[IPL_DATABASE].[dbo].[Batsman].Balls_Played`. The **Seek Predicates** are `Seek Keys[1]: Prefix [IPL_DATABASE].[dbo].[Batsman].Player_ID = Scalar Operator([IPL_DATABASE].[dbo].[Player].[Player_ID])`.

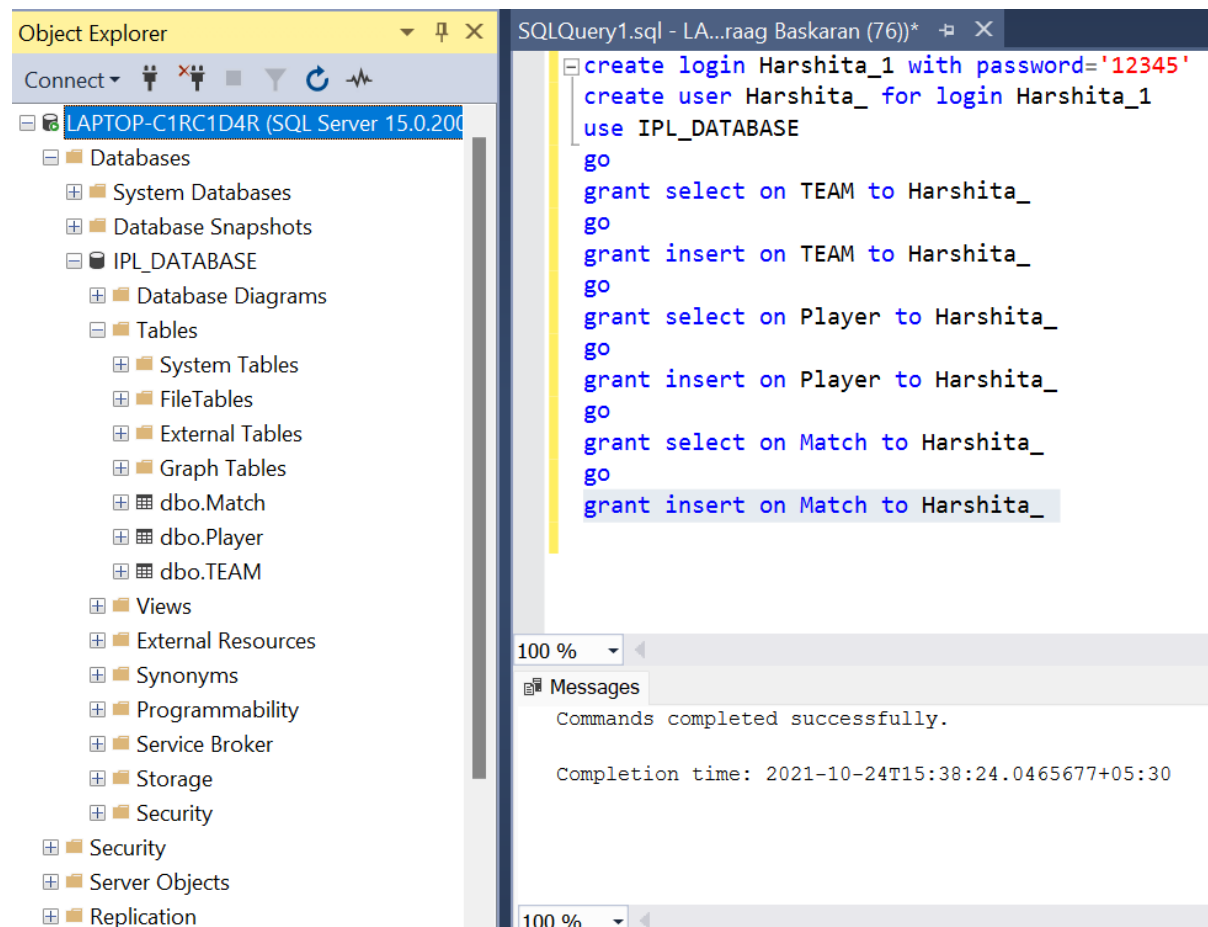
## Users with Different Logins

```
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.

The screenshot displays the SQL Server Enterprise Manager interface. At the top, two query windows are open: 'SQLQuery1.sql - LA...raag Baskaran (76)\*' and 'SQLQuery4.sql - LA...raag Baskaran (57)\*'. The active window shows a SQL script with the following steps:

- `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'`

Below the script, the 'Results' tab is active, showing the output of the queries. The results are as follows:

BeforeTransactionStarts	
1	4

DuringTransaction	
1	0

AfterTransaction	
1	4

The screenshot shows a SQL query editor with two tabs: 'SQLQuery1.sql - LA...raag Baskaran (76))\*' and 'SQLQuery4.sql - LA...raag Baskaran (57))\*'. The active tab contains the following SQL query:

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

Below the editor, there is a toolbar with a zoom dropdown set to '100 %' and two buttons: 'Results' and 'Messages'. The 'Results' button is active, and the results are displayed in a table below:

	SecondSession
1	4

## **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